2009-03

Field programmable gate array (FPGA) based software defined radio (SDR) design

Wright, Durke A.

Monterey, California: Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

FIELD PROGRAMMABLE GATE ARRAY (FPGA) BASED SOFTWARE DEFINED RADIO (SDR) DESIGN

by

Durke Wright

March 2009

Thesis Advisor:	Frank Kragh
Co-Advisor:	Herschel Loomis

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** March 2009	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE Field Programmable Gate Array (FPGA) Based Software Defined Radio (SDR) Design		**5. FUNDING NUMBERS**	
6. AUTHOR(S) Durke Wright			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000		**8. PERFORMING ORGANIZATION REPORT NUMBER**	
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A		**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited		**12b. DISTRIBUTION CODE**	

13. ABSTRACT (maximum 200 words)

There are existing wideband communications systems that were built using field programmable gate array (FPGA)-based software defined radio (SDR) designs. Despite the inherent advantages of these systems, some are functionally restricted by limited output bandwidth. This thesis was conceived in order to mitigate the restrictions imposed on such designs. This was accomplished by designing an FPGA-based SDR that can compress sampled intermediate frequency (IF) signals. The compression scheme used in the final design is based on flexible operator-defined time-frequency bins and independent energy thresholds for each bin. The thesis presents basic design concepts that influenced the development process, the final design implementation created using Xilinx's System Generator software, and the tests used to verify the final design's functional capabilities.

14. SUBJECT TERMS Software Defined Radio, SDR, Field Programmable Gate Array, FPGA, Signal Compression			**15. NUMBER OF PAGES** 127
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

FIELD PROGRAMMABLE GATE ARRAY (FPGA) BASED SOFTWARE DEFINED RADIO (SDR) DESIGN

Durke A. Wright
Lieutenant, United States Navy
B.S., The George Washington University, 2001

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
March 2009

Author: Durke Wright

Approved by: Assistant Professor Frank Kragh
Thesis Advisor

Professor Herschel Loomis
Co-Advisor

Jeffrey B. Knorr
Chairman, Department of Electrical and Computer Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

There are existing wideband communications systems that were built using field programmable gate array (FPGA)-based software defined radio (SDR) designs. Despite the inherent advantages of these systems, some are functionally restricted by limited output bandwidth. This thesis was conceived in order to mitigate the restrictions imposed on such designs. This was accomplished by designing an FPGA-based SDR that can compress sampled intermediate-frequency (IF) signals. The compression scheme used in the final design is based on flexible operator-defined time-frequency bins and independent energy thresholds for each bin. The thesis presents basic design concepts that influenced the development process, the final design implementation created using Xilinx's System Generator software, and the tests used to verify the final design's functional capabilities.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

Government and military organizations rely heavily on the ability to transmit and receive radio signals via communications techniques that incorporate varying degrees of complexity. As a result, advances in communications techniques and digital signal processing (DSP) technology create an ongoing requirement for research and development (R&D), which is both costly and time consuming. These factors contribute to the fact that new radio communications systems often lag behind their operational requirements. In the traditional digital hardware development model, application-specific integrated circuits (ASIC) are designed in order to optimize a radio's performance parameters. The downside of this model is that hardware optimization generally requires replacing hardware, which is both costly and logistically challenging.

The growing field of software defined radio (SDR) addresses some of the limitations imposed by traditional ASIC implementations. SDR designs are developed such that software is used to configure hardware, as needed, to perform different functional operations. This unique capability allows newly developed radios to provide greater technological flexibility. Field programmable gate arrays (FPGAs) have become a key industry component for the development of the hardware logic operations performed in SDR-based systems. As a result, R&D efforts have started to incorporate FPGA-based designs where operationally reasonable. There are existing wideband communications systems that are especially well-suited for these designs. Unfortunately, some of these systems are functionally restricted by relatively limited output bandwidths.

This thesis was conceived to mitigate the restrictions imposed on FPGA-based designs by external input-output (IO) bandwidth mismatches. Efforts were focused on understanding the requirements to design, build, and test an FPGA-based SDR that could process sampled wideband intermediate frequency (IF) signals and compress the input based on reprogrammable parameters. IF signals are often used in communications systems to modulate and process a carrier signal at a frequency lower than the radio frequency (RF) transmission band. The lower IF frequencies allow modulated signals to

be processed at a reduced cost in terms of operational speed and hardware requirements. When appropriate, IF signals can be up-converted to the RF band for transmission or down-converted from the RF band for processing purposes.

The design concept was based on compartmenting sampled IF data into operator-defined time-frequency bins and then comparing the energy in each bin to its independent energy threshold, also specified by the operator. The fast Fourier transform (FFT) served as the central element of the signals analysis process. Based on this framework, signal data related to bins without sufficient energy are discarded. This process resulted in a flexible, automated compression scheme that can more efficiently utilize the output capacity of a system affected by an IO bandwidth mismatch.

The algorithm was designed and tested using Xilinx's System Generator (SysGen) software, which functions as an integrated component of MathWorks' Simulink environment. The tool was also used to synthesize the design, perform the place-and-route functions, and generate the binary file that provides the design's configuration information for the target FPGA. SysGen provided a layer of abstraction that reduced the requirement for programming experience in a hardware description language (HDL). If the design had more stringent timing requirements, the ability to code in a HDL would have been more important.

Two versions of the algorithm were built and tested. The first utilized an 8-point FFT, which simplified analysis efforts. The second version utilized a 1024-point FFT and helped verify the requirements for scaling the design. Both versions were tested using single-frequency and multi-frequency input signals, without restrictions on the output memory. This ensured the basic compression scheme operated properly. The 1024-point algorithm was used to verify the design could automatically adjust its operational mode if available storage capacity became limited. Testing provided valuable insights regarding the effects and workarounds for machine epsilon. Machine epsilon is defined as the smallest positive number that a digital system can recognize and generate. When combined with rounding or truncating, machine epsilon can lead to numerical calculations that should mathematically equal zero but result in other values. If not

properly addressed, this digital error could be catastrophic to a design's operational utility. Testing also verified the design's desired functional operations.

Although the final design operated as expected, it has performance limitations that should be recognized and considered. All of the development tests were conducted such that the digital input frequencies and the defined bin frequencies were an exact match. If frequencies that did not match the FFT window, i.e., frequencies that did not have an integer number of cycles per FFT window, were used there would be a slight smearing effect in the frequency domain. It is assumed that this effect is minor, and would not significantly impact the efficacy of the algorithm. The tests also assumed the input signal and the SDR shared the same sample frequency. If this were not the case, the implementation could correct for this by appropriate interpolation or decimation, or similar multirate signal processing.

Based on the resources available at the beginning of this effort, the SDR was developed for the Xilinx, Virtex-4 FPGA. While this can potentially affect the portability of the design, only one component was used that is not backward compatible to the Xilinx Virtex-1 FPGA. As a result, the workaround requires only minor changes to two affected design components. The research and development process used for this thesis resulted in a simple, FPGA-based signal compressor that can be tailored by operators via parameter settings. The design can facilitate more efficient use of the output capacity available to systems affected by external IO bandwidth mismatches. Despite its simple nature, the SDR and its components can be optimized and used as a platform for future development efforts.

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I dedicate this work to my wife Rhonda. Her support and encouragement were instrumental and have changed my life.

I would also like to express my sincere gratitude to Professor Frank Kragh. His contagious enthusiasm was my first introduction to the SDR concept and his support and guidance were critical throughout the thesis process.

It goes without saying that I appreciate all the professors and staff that helped bring this thesis together but I would like to mention a few: Professor Herschel Loomis; Professor Alan Ross; Professor Roberto Cristi; Professor Alexander Julian; and Donna Miller.

Thank you.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. BACKGROUND

There are government and military organizations that rely heavily on the ability to transmit and receive communications modulations that incorporate varying degrees of complexity. As a result, advances in communications techniques and digital signals processing (DSP) technology create an ongoing requirement for research and development (R&D) that is both costly and time consuming. As mission requirements shift and resources become limited, be it budget constraints or physical space for a new system, prioritization of supported technologies becomes critical.

Regardless of how these decisions are made, when radio communications systems are developed into digital hardware platforms they have traditionally been designed with application-specific integrated circuits (ASIC) [1]. ASIC designs are beneficial in that they are built to optimize a radio's performance parameters such as speed, power consumption, physical size, etc. However, since ASIC hardware is optimized for a specific set of technologies, once a system has moved through the R&D process into production there is limited flexibility to augment its capabilities. Keep in mind that it often takes years to get from a design concept to the production phase [2]. As a result, there is a lag between the generation of an operational requirement and the availability of a viable system. In light of the fact that communications techniques and the capabilities of digital processing systems are constantly changing, the delays and limitations associated with traditional communications hardware development requires special attention.

A growing field within the communications industry referred to as software radio provides an ideal solution for the limitations of ASIC hardware development. As opposed to working with hard-wired electronic components that are designed for specific modulation techniques and data packet structures, software radio designs offer varying degrees of flexibility. Within industry, the term software radio is often used to describe "a radio that is substantially defined in software and whose physical layer behavior can be

significantly altered through changes to its software" [3]. This means that the radio's functional capability can be changed, on demand, based on existing operational requirements.

In comparison to traditional radio development, the physical hardware used for SDR designs requires a more dynamic range of flexibility and capability [3]. Field-programmable gate arrays (FPGA) provide a high degree of flexibility with respect to implementing logical algorithms in hardware. As a result, many new radio development efforts have started to incorporate FPGA-based SDR designs where it is operationally reasonable. There are some existing wideband communications systems that are especially well suited for these designs as they can potentially accommodate a large spectrum of communications techniques. Unfortunately, some are functionality restricted by relatively limited output bandwidths.

B. OBJECTIVE

This thesis was conceived to help mitigate the restrictions imposed on FPGA-based designs by external input-output (IO) bandwidth mismatch. R&D efforts were focused on the requirements to design, build, and test an FPGA-based SDR for a system that can receive wideband intermediate frequency (IF) signals and generate output with only specific portions of the original input signal. IF signals are often used in communications systems to modulate and process carrier signals at a frequency lower then the radio frequency (RF) transmission band. The lower IF frequencies allow modulated signals to be processed at a reduced cost in terms of operational speed and hardware requirements. When appropriate, IF signals can be up-converted to the RF band for transmission or down-converted from the RF band for processing purposes.

Based on the design requirements, the signal compressor's output must be determined in a dynamic, automated fashion in accordance with programmable control parameters and relevant properties of the received signal. In doing so, the SDR will better utilize a wideband system's limited output bandwidth.

C. SDR DESIGN OVERVIEW

The basic design goal was to develop a software defined radio (SDR) that is able to process wideband IF composite signals, compress the signals based upon operator-defined parameters, and store the relevant information for future processing. Figure 1 through Figure 3 provide a graphic representation of the general concept. As a high-level example, Figure 1 depicts the time-frequency distribution of IF energy received by the radio. The boxes in Figure 2 represent three distinct time-frequency bins of interest to an operator. When all bins are considered together, they are referred to as a bin set. Prior to initializing the SDR, operators will define independent thresholds for each bin. Assuming the top two bins in Figure 2 meet or exceed their established thresholds and the bottom bin does not, Figure 3 shows the compressed data set that would be stored for future processing.

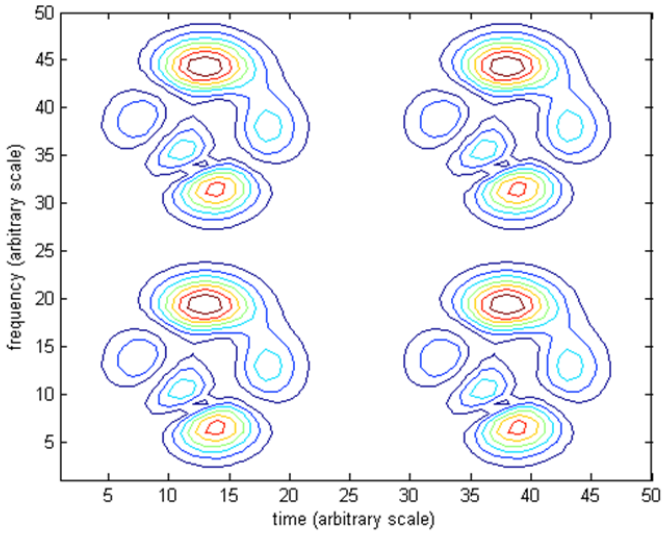


Figure 1 . Received IF Energy (Conceptual).

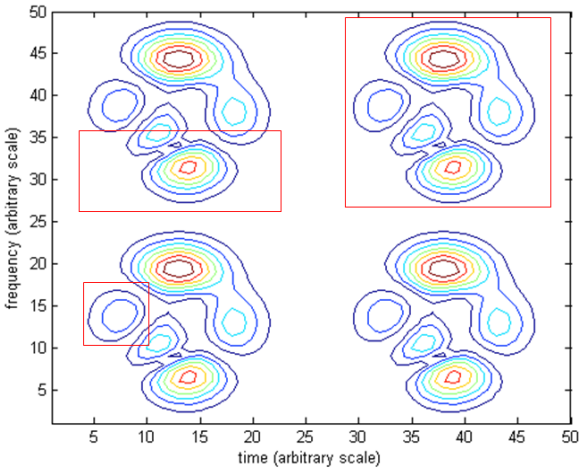


Figure 2 . Signals of Interest (Conceptual).

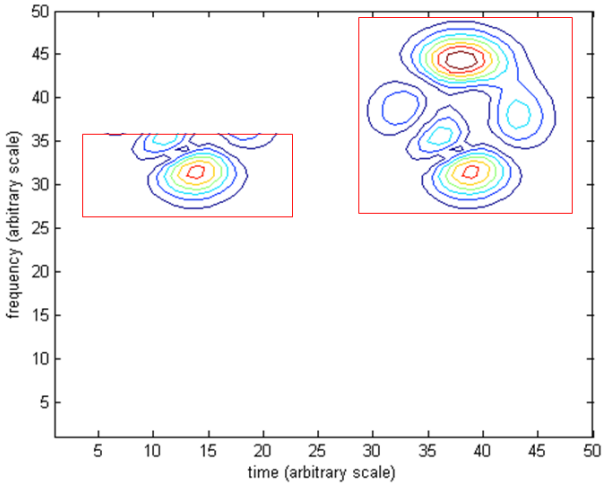


Figure 3 . Compressed Output (Conceptual).

A simple scheme is used to implement the compression mentioned above. The scheme is built around the idea of bins being defined by time and frequency coordinates. Each bin (b) is defined with independent parameters. Using Figure 4 to illustrate the concept, the first bin, $b=1$, is defined between frequencies f_1 and f_2 for a duration of T_1 seconds, while the third bin, $b=3$, is setup to analyze frequencies f_3 through f_4 for a duration of T_3 seconds. The main point is that each bin can be tailored to analyze a

unique set of IF signal characteristics. This flexibility is provided in order to enable tailored detection of different modulation techniques, ranging from burst communications to frequency-hopped transmissions.

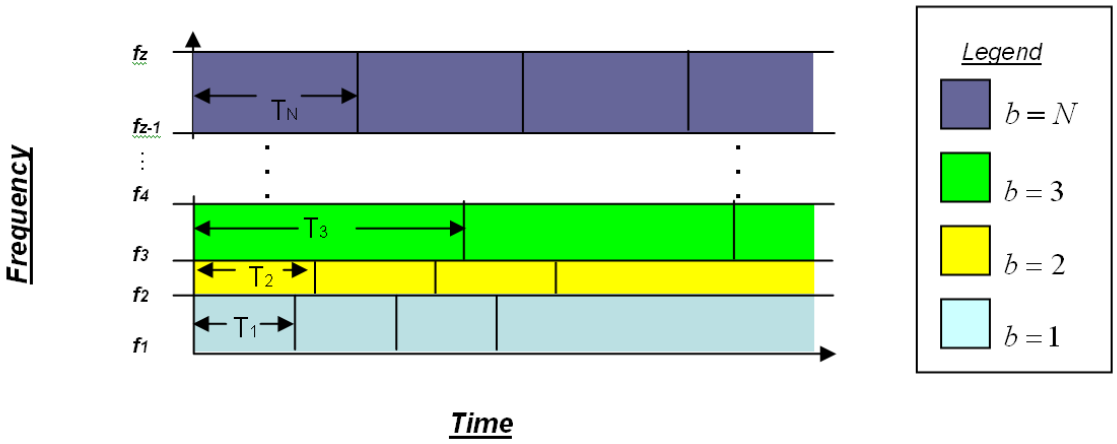


Figure 4 . Bin Dimensions (Conceptual).

In practice, an operator sets frequency and duration parameters for each bin and then establishes independent energy thresholds for each. With this information, the radio is able to process and analyze IF signals and then store information relative to bins that meet or exceed operator-defined thresholds. Information associated with bins that do not meet established thresholds will be discarded. In this way, signal information is compressed.

D. RELATED WORKS

The work done for this thesis is not directly associated with any other efforts, but it does relate to two ongoing areas of research and development: spectral analysis of signal energy and the development of FPGA designs using Xilinx's System Generator software tool.

As outlined in the previous section, the SDR design for this thesis is built on the concept of evaluating the energy in IF signals with respect to time-frequency bins. The time-frequency (TF) construct that was implemented in the final design was based on a simple process that will be explained in Chapter III, but it is worth noting that there are

other well-established algorithms for spectral analysis of time-varying (TV) signals. TV signals are common in real-world applications and have frequency properties that change with time [4]. Some examples include the impulse response of a wireless communications channel, radar and sonar acoustic waves, vocals in speech, engine noise, and jamming interference signals.

Although outside the scope of this thesis, there are numerous methods for analyzing TV signals. As a brief introduction, the two most common methods are the Wigner distribution and spectrograms [4]. However, depending on the target and applied signal properties, other classes of TF analysis may be better suited for providing accurate signal representations. Some of the more commonly referenced classes include, short-time Fourier transforms (STFT), quadratic time-frequency transforms, and wavelet transforms [4,5,6,7,8]. Each class has a set of characteristics that dictate how well it processes different TF signatures. Since most applications are designed to extract specific signatures, this should help guide the choice of analysis methods.

As for Xilinx's System Generator (SysGen) software, a brief description of the tool and its functional application are provided in Chapter II. The major significance of the tool is that it enables a model-based method for developing and testing FPGA designs. This capability greatly reduces the requirement for specialization with the languages and tools traditionally used for FPGA designs [1]. As a result, hardware and software engineers of varying backgrounds now have the ability to utilize the flexibility and parallel computing capacity that are inherent to FPGA hardware platforms [9, 10]. A short, diverse sample of applications that SysGen has been used to develop includes: An FM demodulator [11]; a GPS receiver channel [12]; a reconfigurable video encryption system [13]; a non-coherent frequency shift keying (FSK) transceiver [14]; and a motor incremental shaft encoder [15]. The tool has gained so much popularity that tools and design concepts are being developed by external sources to work with the Xilinx software [16, 17].

E. THESIS ORGANIZATION

This introductory chapter provides brief insight as to the relevance of software defined radios and FPGAs in today's rapidly changing communications environment. The chapter then highlights the effects of external IO bandwidth mismatches on modern communications radios and introduces the design concept used in this thesis to mitigate the associated technical issues. Finally, the chapter highlights time-frequency analysis options and the broad applicability of Xilinx's System Generator software.

Chapter II, Design Considerations, highlights the key concepts and tools used to develop the design. The chapter provides a brief overview of Fourier analysis, FPGAs, FPGA design flow, and the Xilinx System Generator (SysGen) software, which was used as the development environment.

Chapter III, Conceptual Design Model, provides a conceptual description of the modules and data management strategies used to develop the SDR design. The major modules discussed include Bin Energy Calculation, Bin Threshold Analysis, and Data Management.

Chapter IV, Design Implementation Details, focuses on implementation of the conceptual model in the SysGen development environment. Detailed descriptions of each design element are provided in context with its parent modules and any interrelated components.

Chapter V, Design Testing, explains the tests used to validate the designed SDR's functional operations. The chapter then provides test results and analysis.

Chapter VI, Conclusion, summarizes the body of work captured in the thesis and then provides recommendations for future work that could enhance the SDR design.

THIS PAGE INTENTIONALLY LEFT BLANK

II. DESIGN CONSIDERATIONS

In order to design the SDR that was discussed in the previous chapter, a few key concepts and design tools proved to be instrumental. This chapter will provide a brief overview of these elements to include Fourier analysis, FPGA Basics FPGA design flow, and Xilinx's System Generator tool.

A. FOURIER ANALYSIS

When working with electronic signals, it is often necessary to analyze the different frequencies present in the signal. To do so, it is common practice to utilize some form of Fourier analysis to translate the signal from its time domain representation into a frequency domain representation. The Fourier transform (FT) and its inverse (IFT) are ideal algorithms when working with continuous signals. For digitally sampled signals, the discrete Fourier transform (DFT) and the inverse discrete Fourier transform (IDFT) are ideal for working between domains. Definitions for each of the transforms are provided below [18].

The function $g(t)$, where t represents time, will be used in this section as the time domain representation of the signal. The Fourier transform, $G(f)$, of the signal, where f is an analog frequency, is the frequency domain representation of the signal.

$$G(f) = \mathrm{FT}\{g(t)\} = \int_{-\infty}^{\infty} g(t)e^{-j2\pi tf}\,dt \tag{II.1}$$

The inverse Fourier transform can be used to translate back to the time domain.

$$g(t) = \mathrm{IFT}\ \{G(f)\} = \int_{-\infty}^{\infty} G(f)e^{j2\pi tf}\,df \tag{II.2}$$

Since the design for this thesis will work with signals that are sampled at a specific sample frequency (F_s), the signal must also have a sample-based representation, $g[n]$. The functions $g(t)$ and $g[n]$ are related based on the system's sample period, T_s, which is the inverse of the sample frequency.

$$g[n] = g(nT_s) \tag{II.3}$$

To better understand the relationship between $g(t)$ and $g[n]$, refer to Figure 5 [18].

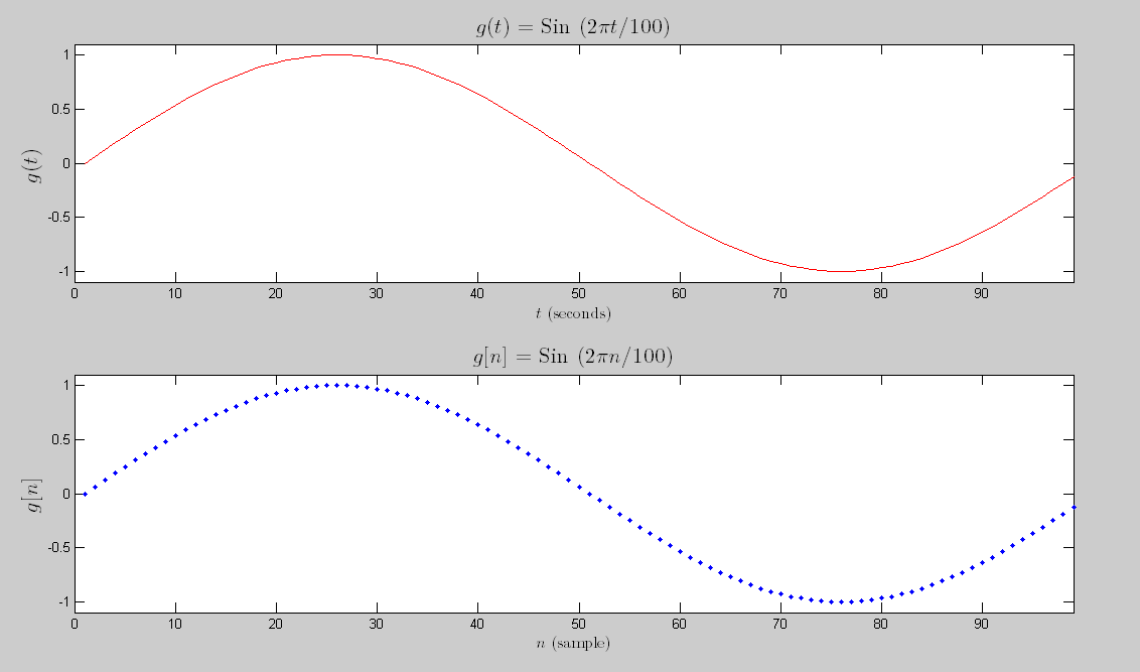


Figure 5 . Time-Based vs. Sample-Based Signals.

The sample-based representation of the signal is used in the DFT algorithm to derive the digital frequency domain representation,

$$G[k] = \frac{1}{N}\sum_{n=0}^{N-1} g[n]e^{-j(2\pi/N)kn} \tag{II.4}$$

where N equals the number of samples in the analysis period and k represents a digital frequency between 0 and $N-1$. The digital frequency can be mapped to an analog frequency based on the sample frequency, and the analysis period, N

$$f = \frac{kF_s}{N} \tag{II.5}$$

The IDFT can be used to translate from the frequency domain back to the sample-based representation,

$$g[n] = \sum_{k=0}^{N-1} G[k]e^{j(2\pi/N)kn} \tag{II.6}$$

where n is between 0 and $N-1$ [18].

For this design, three properties of Fourier analysis need to be highlighted. First, when a signal is sampled in the time domain, its DFT is periodic, with period N. As a result of this repetitive relationship and in accordance with Nyquist's sampling theorem [3], the signal's bandwidth (F_b) must be less than half the sample frequency, F_s.

$$F_b < \frac{F_s}{2} \tag{II.7}$$

Otherwise, the frequency spectrum can not be represented without interference called aliasing. This fact dictates that $F_s/2$ is the highest analog frequency that should be evaluated using the DFT. The second property worth noting relates to the analysis period, N, of the DFT. More samples in an analysis period yields greater frequency resolution. The third property relates to the conjugate symmetry of the DFT. If the sampled signal, $g[n]$, is real-valued, then its DFT has conjugate symmetry, i.e., [18]

$$G[k] = G^*[N-k]. \tag{II.8}$$

Another point worth noting is the DFT as presented in Eq. (II.4) requires N^2 numerical operations for the straightforward calculation. Several algorithms referred to as fast Fourier transforms (FFTs) were created to generate the numerical values of the DFT, but in fewer operations. As opposed to N^2 operations, FFT algorithms require only $N\log(N)$ operations for a period of N samples [18].

B. FPGA BASICS

Field-programmable gate arrays (FPGAs) are semiconductor devices containing logic components and interconnects that are both programmable. Together, these elements can be combined to perform simple gate level logic operations (AND, XOR, etc), or more complex combinational logic functions. Although FPGAs are usually slower and draw more power than ASIC designs [1], the ability to reprogram an FPGA gives the device great versatility and inherent advantages. For instance, it generally takes less time to develop an FPGA based product for the market place [2,10]. FPGAs allow developers to upgrade systems and fix bugs without requiring hardware changes and with less design cost. As an additional benefit, the architecture of an FPGA enables designs to perform multiple computational operations in parallel. Parallelism allows for

considerable data throughput at relatively low clock rates. These characteristics have proven especially useful in the fields of aerospace and defense systems, ASIC prototyping, digital signal processing, and software-defined radio (SDR) [10].

C. FPGA DESIGN FLOW

The design process for working with an FPGA can generally be broken into four basic steps. First, define and verify the functional behavior that the hardware is expected to implement. Next, synthesize the design so the logical description is translated into a structural model. Once synthesis is complete, implement a process called place-and-route and generate a binary (bin) file, which contains the configuration information that will be loaded onto the physical FPGA platform. Finally, load the bin file onto the target platform and test the design using physical signals, as opposed to computer generated stimulus [19,20,21].

1. Define and Verify Functional (Logical) Behavior

In simple terms, an FPGA's functional behavior is a model that defines the system's high-level logical operation. This model defines system inputs, the process and order for working with the relevant signals, and system outputs. It is important to note that this high-level model should consider all relevant requirements such as physical constraints, performance, interface, cost, power, etc. Once all these elements are put into the proper perspective, designers can effectively define and test the logical operations of their desired system.

In order to define functional behavior, designers select the best suited logical components, or build them if required. Then component connections are established in conjunction with a meaningful order of operations. There are a variety of tools available for working through this process, but they all must capture the design parameters using a hardware description language (HDL). It is worth noting that there are two industry standard HDLs: Verilog and Very High Speed Integrated Circuit (VHSIC) Hardware Description Language (VHDL). There are advantages to each, but that discussion is outside the scope of this thesis. Regardless of which language or tools are used to define

functional behavior, it is important to ensure all the intended logical operations are performed. Therefore, as the system is being designed, it is wise to build and run simulations as an iterative process.

2. Synthesis

Synthesis is a two-step process that takes a high-level behavioral model and maps it according to the design's established interconnections. There are a variety of synthesis tools on the market, but their principal functions are all the same. The first step in the synthesis process involves verification of code syntax used in the behavioral model. Once the syntax is deemed correct, the functional description is translated to a structural model. This new model is captured in the form of a netlist, which defines connections and constraints between the components in an electronic design.

3. Place-and-Route (PAR)

The next step toward the physical implementation of the design is referred to as Place-and-Route (PAR). During the PAR phase, connections outlined in the netlist are translated into gate level logic that consisting of lookup table (LUTs), flip flops (FF), memory blocks, and input/output (IO) modules. This process is directly correlated with the target FPGA. Point being, if the same netlist were used in multiple FPGA types, the PAR results would be unique for each. The translation of connections serves as a bridge between the logical design and the physical implementation. The output of the PAR process is used to generate the .bin file that is loaded onto the target FPGA.

4. Testing

Once the .bin file is loaded onto the target platform, the final step in the design process involves verifying the FPGA correctly implements the configured operations within the required specifications.

D. SYSTEM GENERATOR (SYSGEN)

SysGen is a schematic based design tool that enables graphic modeling and simulation of FPGA behavior. The software suite can also be used for synthesis, PAR, and generation of the binary file that is loaded onto the target FPGA board. The software suite was designed to work seamlessly within MathWorks' Simulink modeling infrastructure. SysGen provides libraries of optimized Intellectual Property (IP) cores that can be graphically connected and configured. An IP core is a reusable block of code, often a generic netlist that protects a vendor against reverse-engineering. Throughout this paper, System Generator's IP cores are referred to as SysGen modules. The SysGen libraries are integrated into the Simulink library. When a library is selected, the SysGen sublibrary icons are clearly distinguishable by an outlined 'X' icon that indicates "Xilinx."

In a fashion similar to Simulink, all SysGen modules can be configured by double clicking the associated icon and updating the appropriate parameters. A major distinction for SysGen modules is that each has HDL code embedded in its definition. This minimizes the developer's requirement to generate code for each independent component. This layer of abstraction can reduce development time and the learning curve required for basic systems design. If a necessary component does not exist in the SysGen library, it is possible to develop a software algorithm using MATLAB code (M-code) and later use SysGen's functional capability to translate the code into the desired HDL format (VHDL or Verilog). This functionality is enabled by the SysGen M-code module shown in Figure 6. Once an M-code algorithm is developed and associated with a SysGen M-code module, it performs its designed logical behavior in concert with the other SysGen modules in a design. Despite the obvious benefits of the SysGen suite, Xilinx does not recommend its use as a complete replacement for HDL coding. They encourage hands-on coding for parts of a design that require management of internal hardware clocks [20].

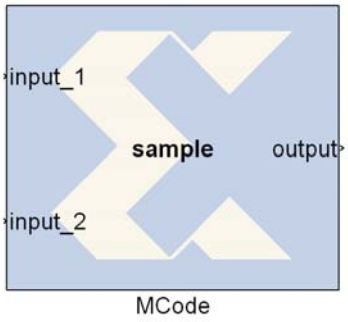


Figure 6 . SysGen M-code Module.

As a visual overview of how the SysGen suite works, observe Figure 7. The red box in the upper left hand corner contains a model designed using Simulink components. The system multiplies a sine wave by a random number and then adds a constant to the product. The final output signal can then be viewed graphically using the Simulink scope.

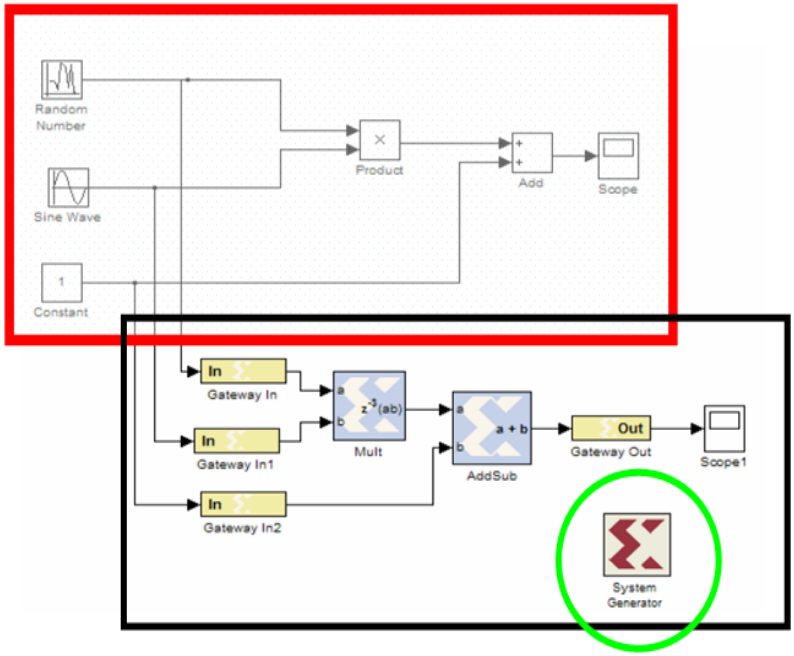


Figure 7 . Simulink to SysGen Comparison.

The system contained in the black box is functionally identical to the Simulink model, but it is built using SysGen modules. The only major difference is that the System Generator model requires Gateway modules to signify FPGA IO ports. It also requires the SysGen System Generator module, circled in green, which is used to initialize the model and specify how code generation and simulation should be handled. Assuming all other parameters in the two models match, their simulation results would be identical. This commonality allows SysGen simulations to be compared to the bench mark results generated from a Simulink models. It can also reduce the effort normally required to write test benches for HDL designs.

Once a SysGen design's functional behavior has been tested to satisfaction, the software suite can then generate several useful system level outputs to include HDL, a netlist, or even a .bin file. To specify the desired output, a designer simply selects the appropriate format from the 'Compilation' parameter in the SysGen System Generator module's interface as illustrated in Figure 8. If an HDL Netlist were chosen, as shown in the figure, the software would generate an HDL file that could be utilized in a variety of development tools. For this thesis, SysGen was used to generate a .bin file so the compilation parameter was set to 'Bitstream'.

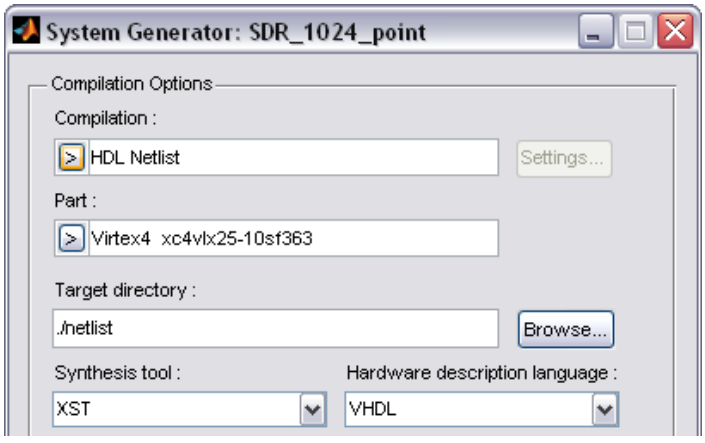


Figure 8 . SysGen System Generator Module Interface.

The SysGen System Generator module is also used to specify other critical design parameters such as FPGA chipset, desired HDL format, clock rate, and more. Once all the required values are specified, the desired output file is produced by clicking the 'Generate' button located in the lower left-hand corner of the System Generator module's interface. The specified output file format is then created and stored in the Target directory specified within the interface.

E. SUMMARY

This chapter highlights the concepts that were relevant to the SDR design process: Fourier analysis, FPGAs, the FPGA design flow, and Xilinx's System Generator (SysGen). Of all the topics discussed, the fast Fourier transform (FFT) and SysGen software tool are the two most central elements of this design's overall development process. The next chapter provides a conceptual description of the modules and data management strategies used to develop the design.

THIS PAGE INTENTIONALLY LEFT BLANK

III. CONCEPTUAL DESIGN MODEL

In order to develop the SDR outlined in the first chapter, a design was built around the conceptual model shown in Figure 9.

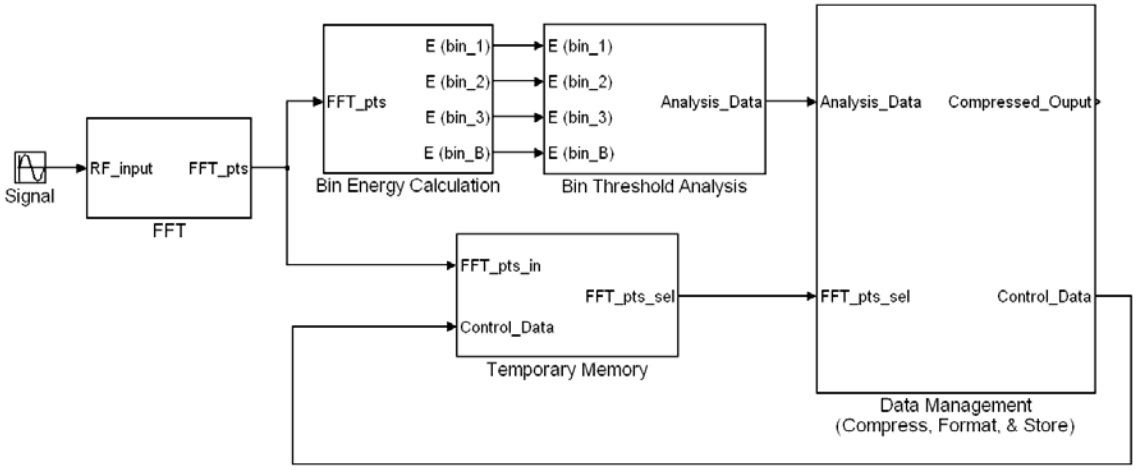


Figure 9 . SDR Modules. (Conceptual).

Each module's basic functionality is outlined below, but more detailed descriptions are provided in Sections III.A through III.E:

- Signal: A pre-demodulated (pre-D) IF signal is applied to the system input
- FFT: The IF input signal is processed through a fast Fourier transform (FFT) so that it can be analyzed in the frequency domain
- Bin Energy Calculation: The input signal's frequency domain information is associated with the appropriate operator-defined bins. Once all the data points for each bin are processed, energy in each bin is calculated for use in the Bin Threshold Analysis module. Operators will also have the ability to prioritize the relative importance of each individual bin. This will be useful in the final Data Management module.

Note: FFT signal data are stored in temporary memory for use in later stages of the design.

- Bin Threshold Analysis: First, each bin is analyzed and its energy is compared to its established threshold. Then, control data pertaining to bins with sufficient energy are generated and passed to the Data Management module.
- Data Management: Control data are used to manage four interrelated processes:
  - Read FFT signal data from temporary memory ;
  - Compress the dataset so it only includes information associated with bins that meet threshold requirements;
  - Store the compressed dataset in final output memory;
  - Manage available output memory.

The following sections cover the general strategies used to design the modules outlined above.

A. SIGNAL

For testing purposes, IF input signals are generated digitally and passed through the system. Sinusoids of known frequencies are used as examples throughout this paper.

B. FFT

The FFT module is the first interface between the IF input signal and the rest of the design. The output of the module consists of both real (X) and imaginary (Y) components. For a given output sample, both components share the same FFT index, k , which will always fall between the values of zero and one less than the number of points per FFT (N).

$$0 \le k \le N-1 \qquad \text{(III.1)}$$

Although the concepts will make more sense after reading the upcoming sections on windowing, it is important to know that all FFT data points are associated with both time and frequency windows. To manage these relationships, FFT data are stored in multiple vectors and the notational descriptions are provided in the relevant sections.

C. BIN ENERGY CALCULATION

In order to analyze the bins as depicted in Figure 4, it is necessary to examine the IF input in terms of its frequency and time components. First, input signals are passed through an N-point FFT for analysis in the frequency domain. All FFT data are stored in temporary memory for processing in a later stage. Once in the frequency domain, signal energy can be calculated with respect to both time and frequency windows. Figure 10 shows the conceptual model used to calculate energy values.

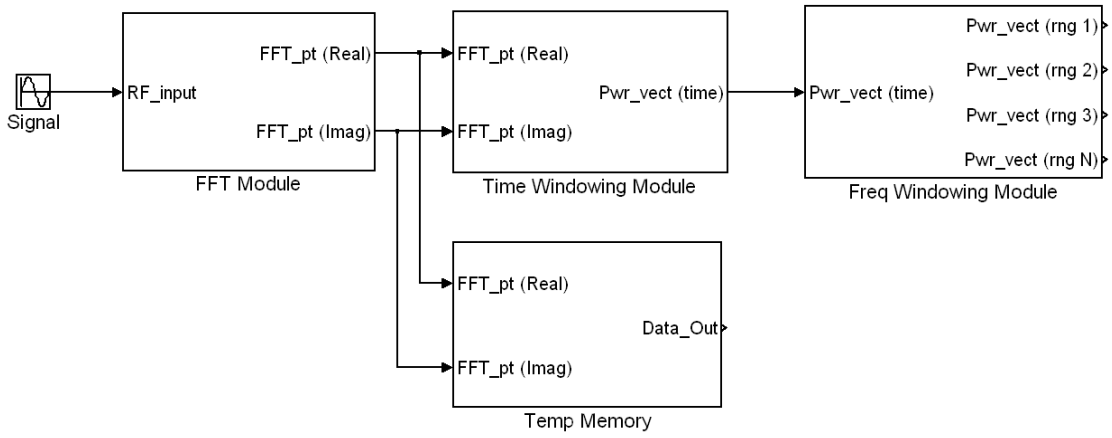


Figure 10 . Bin Energy Calculation Module (Conceptual).

1. Time Windowing

A few things should be noted about the relationship between sample frequency, F_s, and time windowing for this design. First, the F_s of the input signal and FFT module are assumed to be equal. This means that the smallest continuous block of time, or minimum time-window, that can be analyzed (T_{min}) is a function of the FFT period, N, and the sample time ($T_s = 1/F_s$),

$$T_{min} = NT_s = \frac{N}{F_s} \tag{III.2}$$

This also means that for this design, an operator-defined time-window period (T) must be an integer (M) multiple of T_{min},

$$T = MT_{min} = \frac{MN}{F_s} \tag{III.3}$$

As an example of how this works, if $T_s = 3\mu s$ and $N = 8$ then $T_{min} = 8$ x 3µs =24µs . Therefore, all time-windows must be multiples of 24 µs (i.e., T must be 24, 48, 72, etc.).

Figure 11 gives a visual example of how the process works using $M = 3$ to determine T. The input signal shown in figure's the top graph,

$$\sin\left(2\pi\frac{n}{8}\right), \text{ where } n = [0,1,2,..39] \tag{III.4}$$

will be used as the reference input throughout this thesis. It has a period of 8 samples, so to simplify analysis an 8-point FFT is utilized.

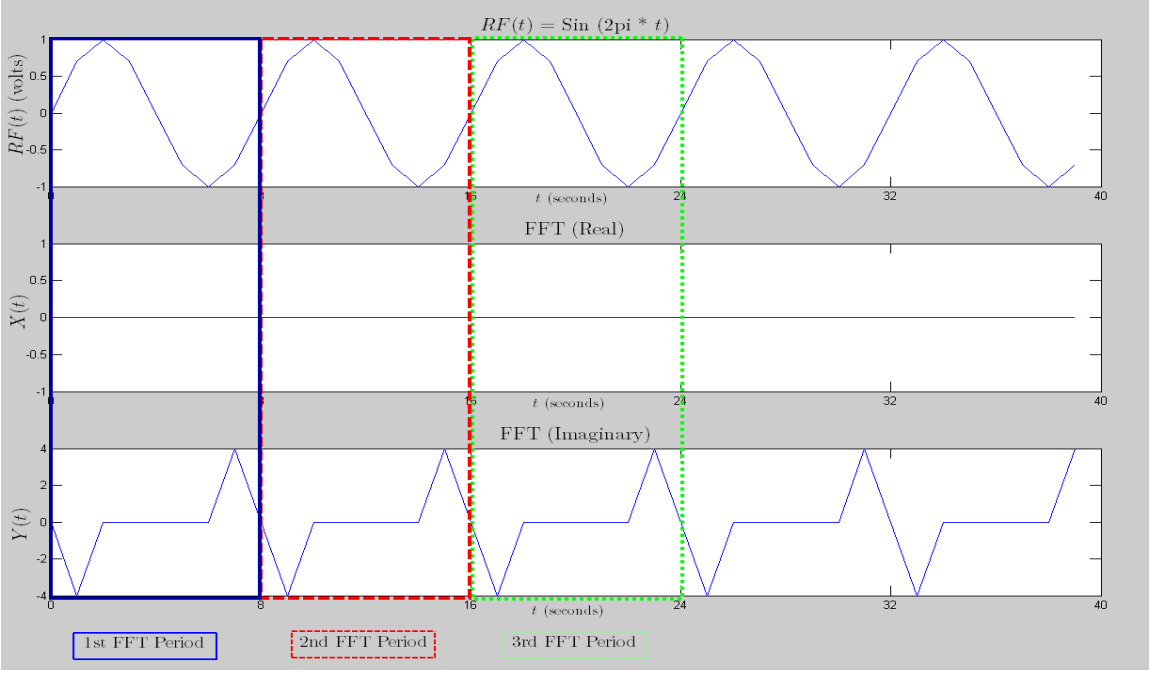


Figure 11. Input Signal Compared to FFT Analysis.

This method of time windowing is used to facilitate the design's energy calculation process. The first step involves calculating the energy at each FFT index (k). The is done by adding the squared FFT values of the real, $X(k)$, and imaginary, $Y(k)$, index components and storing the value in $E(k)$,

$$E(k) = X^2(k) + Y^2(k) \tag{III.5}$$

Based on the input signal in Figure 11, the real component of the FFT is zero, so it will not be mentioned in later analysis. Figure 12 shows the general concept covered in Eq. (III.5).

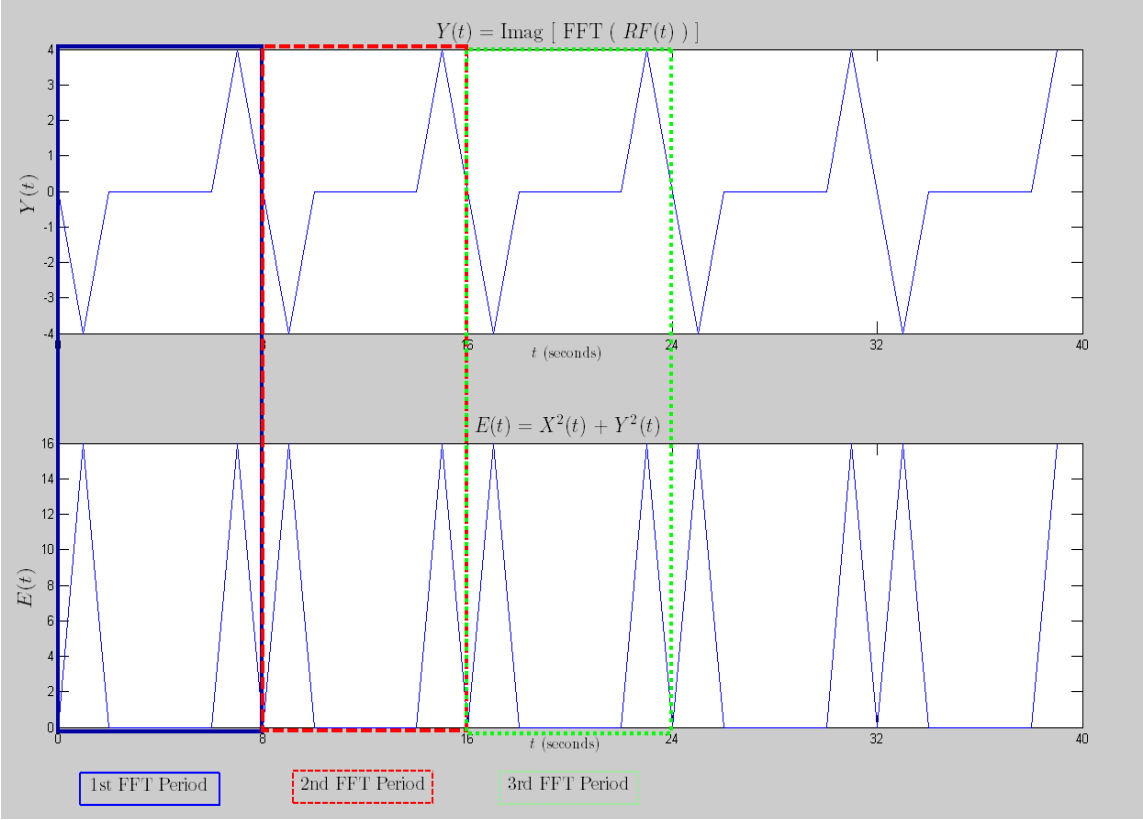


Figure 12. FFT Data Converted to Energy.

Each $E(k)$ value is stored in memory so that it can be appended to a vector that represents the energy values, $E_{min}(m)$, in an FFT period,

$$\mathbf{E}_{min}(m) = [E(0), E(2), \ldots, E(N-1)] \tag{III.6}$$

The index m indicates which FFT period is being processed. These vectors are stored in memory so the energy in the operator-defined time window, $\mathbf{E}_{time}(w)$, can be calculated.

$$\mathbf{E}_{time}(w) = \sum_{m=1}^{M} \mathbf{E}_{min}(m) \tag{III.7}$$

The index w indicates which operator-defined time window sequence being processed. The notation $\mathbf{E}_{time}(w,k)$ is used to reference the k^{th} index within the vector. Each w index is associated with a set of m indices, $\mathbf{m}(w)$, and the relationship is a function of M, the number of FFT periods in a time-window

$$\mathbf{m}(w) = [(w-1)M + 1, \ \ldots, wM] \tag{III.8}$$

The notation $\mathbf{m}(w,k)$ is used to reference the k^{th} component of the vector.

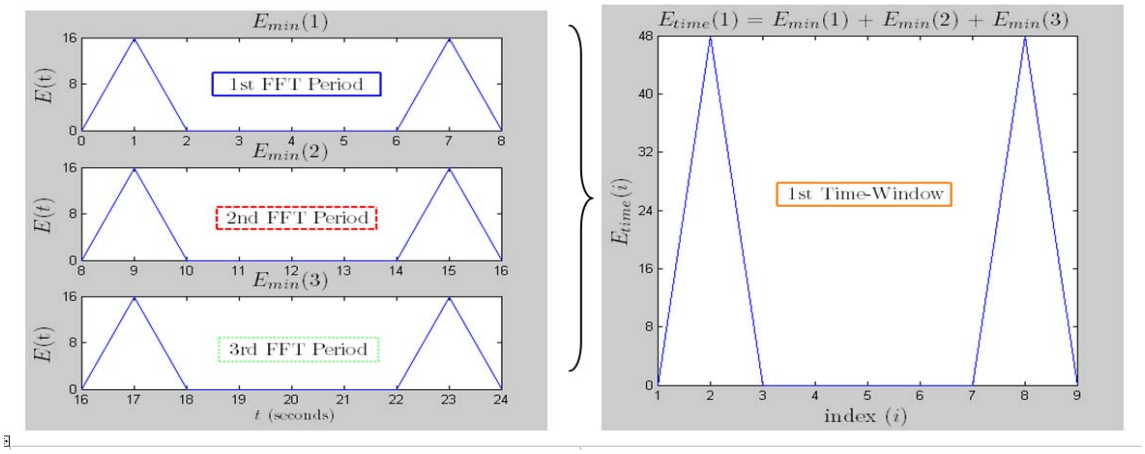
Equaton (III.7) represents a vector-based accumulation of the $\mathbf{E}_{min}(m)$ values. Analysis of the input signal in Eq. (III.4) will be used to illustrate the $\mathbf{E}_{time}(w)$ calculation process.

- Energy during FFT periods:

  Based on the bottom graph in Figure 12, the energy spectrum for each FFT period is equal, so the associated $\mathbf{E}_{min}(m)$ vectors are also equal. As the data in the figure illustrates, $\mathbf{E}_{min}(1) = \mathbf{E}_{min}(2) = \mathbf{E}_{min}(3) = [0,16,0,0,0,0,0,16]$.

- $\mathbf{E}_{time}(w)$ energy calculation:

  In accordance with Eq. (III.6), the values in each $\mathbf{E}_{min}(m)$ vector are added by index. As a result, $\mathbf{E}_{time}(1) = \sum_{m=1}^{3} \mathbf{E}_{min}(m) = [0,48,0,0,0,0,0,48]$. Figure 13 is provided as a visual example of the process.



(Energy Calculations) (Vector-Based Accumulation)

Figure 13. Time Windowing Process.

In order to reference the k^{th} component in an $E_{time}(w)$ vector, the notation $E_{time}(w,k)$ will be used. The vector will always have N total indices starting at zero (analogous to FFT indices).

Referring back to the overview in Section III.B, FFT data values from a specific FFT period are stored in $\mathbf{X}_{min}(m)$ and $\mathbf{Y}_{min}(m)$ vectors,

$$\mathbf{X}_{min}(m) = [X(1)\ ,\ X(2)\ ,\ \ldots\ ,\ X(N)] \tag{III.9}$$

$$\mathbf{Y}_{min}(m) = [Y(1)\ ,\ Y(2)\ ,\ \ldots\ ,\ \mathrm{Y}(N)] \tag{III.10}$$

In order to reference the k^{th} component in either vector, the notation $\mathbf{X}(m,k)$ and $\mathbf{Y}(m,k)$ will be used. These vectors are then used to generate vectors $\mathbf{X}_{time}(w)$ and $\mathbf{Y}_{time}(w)$, which represent time-windows

$$\mathbf{X}_{time}(w) = \left[\mathbf{X}_{min}\left((w\text{-}1)M+1\right),\ \mathbf{X}_{min}\left((w-1)\mathrm{M}\text{+}2\right),\ \ldots\ ,\ \mathbf{X}_{min}\left(wM\right)\right] \tag{III.11}$$

$$\mathbf{Y}_{time}(w) = \left[\mathbf{Y}_{min}\left((w\text{-}1)M+1\right),\ \mathbf{Y}_{min}\left((w-1)\mathrm{M}\text{+}2\right),\ \ldots\ ,\ \mathbf{Y}_{min}\left(wM\right)\right] \tag{III.12}$$

In order to reference the q^{th} component in either vector, the notation $\mathbf{X}_{time}(m,q)$ and $\mathbf{Y}_{time}(m,q)$ will be used where

$$1 \le q \le NM\ . \tag{III.13}$$

2. Frequency Windowing

Once the energy for a time window is calculated, it is possible to determine the energy in frequency ranges of interest, which equate to energy in bins (E_{bin}). The 8-point FFT analysis above will be used to illustrate the process of frequency-based calculations. Recall from Section II.A that an 8-point FFT length dictates that $k = 4$ is the maximum digital frequency that can be detected unambiguously. Likewise, if the digital frequency of interest is $k = 1$, then values captured in indices 1 and 7 provide relevant information. Since the SDR's IF input signal is a real signal, the FFT is conjugate symmetric so the FFT values for $k \geq 1 + N/2 = 5$ are redundant and can be disregarded. As a result, FFT indices between zero and half the FFT period, $N/2$, are used to manage energy calculations.

First the operator defines the digital start and stop frequencies for each range of interest (ROI) and the values are stored in the vector, $\mathbf{ROI}(b)$. Next, the FFT indices associated with a ROI are stored in a vector, $\mathbf{L}(b)$. The notation $\mathbf{L}(b,l)$ is used for the l^{th} component in the vector. The variable l represents an index in the $\mathbf{L}(b)$ vector, which can range from one to the total number of indices associated with the bin, $l_{\max}$. Finally, the values in $\mathbf{E}_{time}(w)$ that coincide with the indices in $\mathbf{L}(b)$ are added by index. The resulting sum is equal to the total energy within a frequency range and equates to the energy in the bin, $E_{bin}(w,b)$

$$E_{bin}(w,b) = \sum_{l=1}^{l_{\max}} \mathbf{E}_{time}(w, \mathbf{L}(b,l)) \tag{III.14}$$

Unlike Eq. (III.7) which is vector addition Eq. (III.14) is defined as the sum of the components within a single $E_{time}(w)$ vector, which results in a scalar. For example, assume an operator is interested in two digital frequency ranges such that $\mathbf{ROI}(\mathbf{1}) = [1:2]$ and $\mathbf{ROI}(2) = [3]$. Since the IF input signals are real signals, the resulting FFT data is conjugate-symmetric. As a result, when working with an 8-point

FFT $\mathbf{L}(1)=[1,2]$ and $\mathbf{L}(2)=[3]$. Based on the $\mathbf{E}_{time}(1)$ vector calculated in Section III.C.1, $E_{bin}(1,1)=\sum_{l=1}^{2}\mathbf{E}_{time}(1,\mathbf{L}(1,l))=48+0=48$ and $E_{bin}(1,2)=\sum_{l=1}\mathbf{E}_{time}(1,\mathbf{L}(2,l))=0$.

D. BIN THRESHOLD ANALYSIS

Once the bin set has been fully processed, energy values for each bin are compared to their operator-defined thresholds, $H(b)$. If any of the bins meet or exceed their threshold, then the time window index, w, and other relevant control data are stored. A complete list of control data are provided in Table 1. Once all the bins have been evaluated, an analysis flag is generated to indicate that a bin set's data are ready for compression and storage in the Data Management Module.

Table 1. Control Signal: Threshold Analysis.

Control Data	**Purpose**
anal _ fl	Flag indicates that a bin set has been processed and the dataset is ready for compression
anal_qty(w)	Vector containing the number of bins that meet or exceed their threshold
anal_ROI(*w*)	Vector containing the *ROIs* that meet or exceeds their threshold

To better understand how the Threshold Analysis Module works, the $E_{bin}(w,b)$ example from the previous section is continued. Assume the bin thresholds are $H(1)=43$ and $H(2)=50$:

- Bin energy comparisons

  In the previous example, the energy for the first bin was calculated such that $E_{bin}(1,1)=48$ and the energy in the second bin was calculated such that $E_{bin}(1,2)=0$. Based on the calculated energies and thresholds for each bin $E_{bin}(1,1)<H(1)$ and $E_{bin}(1,2)<H(2)$

- Generated control data

  Since only the first bin met threshold requirements when all bin set data has been processed and is ready for compression, which is indicated by the $anal\_fl$ control signal, data will only be stored for the first bin. Since this example involves the first bin set the associated index is set to one, $w = 1$. The number of passing bins for the bin set is captured with the $anal\_qty(w)$ vector, so $anal\_qty(1) = 1$. In similar fashion, the passing bin(s) are captured in the $\mathbf{anal\_ROI}(w)$ vector, so $\mathbf{anal\_ROI}(1) = 1$.

E. DATA MANAGEMENT

When the $anal\_fl$ flag is set to indicate that the dataset is ready for compression, the data management module generates a signal to control the process for reading FFT signal data out of temporary memory. There are $2N$ data points per FFT period (N real and N imaginary). However, as described in Section III.C.2, the SDR is designed to work with IF signals, which are real in the time domain and conjugate-symmetric in the frequency domain. This symmetry allows the design to disregard the redundant data so that only $2\left[(N/2)+1\right]$ data points per FFT period are utilized, $(N/2)+1$ values being real and the other $(N/2)+1$ are imaginary. Since there are a total of M FFT periods per time-window a total of $2M\left[(N/2)+1\right]$ data points could potentially be written to output memory. As a visual reminder of what this means, refer to Figure 11 and observe the FFT points associated with the three T_{min} periods that comprise the first time-window.

The primary goal of this design is to reduce the number of data points stored in output memory. So, instead of storing $2M\left[(N/2)+1\right]$ data points every T seconds, this module will only store values associated with indices in $ROIs$ that meet defined thresholds. The bin index, b, for each passing ROI is stored in the $\mathbf{anal\_ROI}(w)$ vector. To simplify the explanation, the general concept will only be described with

respect to the imaginary data points stored in the vector associated with the time-window period, $\mathbf{Y}_{time}(w)$. As described above, each range has an $\mathbf{L}$ vector that contains its relevant FFT indices. The vectors for each *ROI* in **anal_ROI**(w) are used to determine the imaginary values that will be stored in the final compressed dataset. This process is broken into two principle steps. First, compile all FFT values associated with qualifying bins into $\mathbf{Y}_{bin}$ vectors,

$$\begin{aligned}\mathbf{Y}_{bin}(w,b) &= \mathbf{Y}_{bin}\left(\left[1,\ \ldots,l_{max},\ (l_{max}+1),\ \ldots,\ 2l_{max},\ \ldots,\ (M-1)l_{max},\ \ldots,\ Ml_{max}\right]\right)\\ &= \mathbf{Y}_{time}\left(w,\mathbf{L}(b,1)\right),\ \ldots,\mathbf{Y}_{time}\left(w,\mathbf{L}\left(b,l_{max}\right)\right),\ \ldots\\ &\ldots\mathbf{Y}_{time}\left(w,\left[N+\mathbf{L}(b,1)\right]\right),\ \ldots,\mathbf{Y}_{time}w,\left[N+\mathbf{L}\left(b,l_{max}\right)\right],\ \ldots,\ \ldots\\ &\ldots\mathbf{Y}_{time}\left(w,\left[(M-1)N+\mathbf{L}(b,1)\right]\right),\ \ldots,\mathbf{Y}_{time}\left(w,\left[(M-1)N+\mathbf{L}\left(b,l_{max}\right)\right]\right)\end{aligned} \tag{III.15}$$

For *ROIs* not included in **anal_ROI**(w), null vectors are assumed. The second step involves combining all $\mathbf{Y}_{bin}(w,b)$ vectors to produce the final imaginary output vector, $\mathbf{Y}_{out}(w)$, for the bin set,

$$\mathbf{Y}_{out}(w)=[\mathbf{Y}_{bin}(w,1),\mathbf{Y}_{bin}(w,2),...,\mathbf{Y}_{bin}(w,s)] \tag{III.16}$$

The variable s indicates the total number of bins that are evaluated in a bin set. The exact same process applies to the real component of the signal data, but the resulting vectors are designated $\mathbf{X}_{bin}(w,b)$ and $\mathbf{X}_{out}(w)$.

$$\begin{aligned}\mathbf{X}_{bin}(w,b) &= \mathbf{X}_{bin}\left(\left[1,\ \ldots,l_{max},\ (l_{max}+1),\ \ldots,\ 2l_{max},\ \ldots,\ (M-1)l_{max},\ \ldots,\ Ml_{max}\right]\right)\\ &= \mathbf{X}_{time}\left(w,\mathbf{L}(b,1)\right),\ \ldots,\mathbf{Y}_{time}\left(w,\mathbf{L}\left(b,l_{max}\right)\right),\ \ldots\\ &\ldots\mathbf{X}_{time}\left(w,\left[N+\mathbf{L}(b,1)\right]\right),\ \ldots,\mathbf{Y}_{time}w,\left[N+\mathbf{L}\left(b,l_{max}\right)\right],\ \ldots,\ \ldots\\ &\ldots\mathbf{X}_{time}\left(w,\left[(M-1)N+\mathbf{L}(b,1)\right]\right),\ \ldots,\mathbf{Y}_{time}\left(w,\left[(M-1)N+\mathbf{L}\left(b,l_{max}\right)\right]\right)\end{aligned} \tag{III.17}$$

$$\mathbf{X}_{out}(w)=[\mathbf{X}_{bin}(w,1),\mathbf{X}_{bin}(w,2),...,\mathbf{X}_{bin}(w,s)] \tag{III.18}$$

A brief example of how $\mathbf{Y}_{out}(w)$ is generated will be explained below based on the input signal and analysis data generated to this point:

- Imaginary FFT values:

  Based on the example in Section III.C.1 and Figure 13 the imaginary values for the first three FFT periods, $\mathbf{Y}_{min}(m)$, are all equal. Therefore $\mathbf{Y}_{min}(1) = \mathbf{Y}_{min}(2) = \mathbf{Y}_{min}(3) = [0,4,0,0,0,0,0,4]$.

- Bin definition and data compression :

  Based on the example in Section III.C.2, the first bin was defined such that it includes the 2nd and 3rd indices of each FFT, which means $\mathbf{L}(1) = [1,2]$. Therefore, in accordance with Eq. (III.15), the vector that contains the compressed data set for the first bin, $\mathbf{Y}_{bin}(1,1)$, must contain the 2nd and 3rd values from each of the $\mathbf{Y}_{min}(m)$ vectors ➔ $\mathbf{Y}_{bin}(1,1) = [4,0,4,0,4,0]$

- Final Output

  Since the first bin is the only one to meet its threshold requirement, it is also the only bin represented in the final output data set, $\mathbf{Y}_{out}(1)$. As a result

  $\mathbf{Y}_{out}(1) = \mathbf{Y}_{bin}(1,1) = [4,0,4,0,4,0]$.

An important point to take from this example is that instead of 24 imaginary FFT values for a time-window period of T seconds, only 6 values will be stored in memory. Therefore compression is achieved.

Without additional data, $\mathbf{X}_{out}(w)$ and $\mathbf{Y}_{out}(w)$ do not provide enough information for an operator to analyze the system's output data. To remove all ambiguity, the Data Management Module generates header data that is stored in the $\mathbf{hdr}(w)$ vector.

$$\mathbf{hdr}(w) = [w\ ,\ anal\_qty(w)\ ,\ pri\_fl(w)\ ,\ \mathbf{anal\_ROI}(w)] \quad \text{(III.19)}$$

The $pri\_fl(w)$ signal is described in more detail in Section IV.B.3.f, but in general terms it indicates the design's operating mode during a bin set's analysis period.

The vector $\mathbf{hdr}(w)$ is prepended to the two data vectors $\mathbf{X}_{out}(w)$ and $\mathbf{Y}_{out}(w)$ to create a more relevant output vector, $\mathbf{Out}(w)$, which is then stored in final output memory.

$$\mathbf{Out}(w) = [\mathbf{hdr}(w)\ ,\ \mathbf{X}_{out}(w)\ ,\ \mathbf{Y}_{out}(w)] \tag{III.20}$$

If available output memory were to become an issue, the process for compressing signal data can be altered to reduce expected storage requirements. This is accomplished by limiting the maximum number of bins per bin set that can be stored in final output memory. Instead of saving data for all s operator-defined bins, the operator provides a secondary constant (s_{pri}) to indicate the alternate maximum number of bins that can be stored. The system will be aware of this situation because onboard memory provides a signal that is used to generate the $pri\_fl(w)$ flag once it reaches a specified capacity. In order to alleviate the problem, the design essentially disregards the lowest priority ranges of interest, even if they meet established requirements. In practice, that means a bin set's final output vector will only include the highest priority signal data. However, the output header will still indicate that other *ROIs* met or exceeded their thresholds.

Below, is a more detailed example of how $\mathbf{Out}(w)$ is generated. Assume the bin set has been fully processed and the resulting analysis data is as follows:

- The SDR processed the 7th time-window ➔ $w = 7$
- Calculated bin energies: $E_{bin}(1) = 15; E_{bin}(2) = 50; E_{bin}(3) = 46$
- Operator defined bin thresholds: $H(1) = 10; H(2) = 45; H(3) = 40$
- Bin set data (Real):

  $\mathbf{X}_{bin}(7,1) = [0,1]$, $\mathbf{X}_{bin}(7,2) = [2,3,4,5]$, and $\mathbf{X}_{bin}(7,3) = [6,7,8]$
- Bin set data (Imaginary):

  $\mathbf{Y}_{bin}(7,1) = [1,2]$, $\mathbf{Y}_{bin}(7,2) = [3,4,5,6]$, and $\mathbf{Y}_{bin}(7,3) = [7,8,9]$
- Memory shortage indicator is set ➔ $pri\_fl(7) = 1$

- In a memory shortage condition, the number of bins to store per bin set is set to 1 ➔ $s_{pri} = 1$

Recall:

$$\mathbf{hdr}(w) = \left[w, anal\_qty(w), pri\_fl(w), \mathbf{anal\_ROI}(w) \right]$$
$$\mathbf{Out}(w) = \left[\mathbf{hdr}(w), \mathbf{X}_{out}(w), \mathbf{Y}_{out}(w) \right]$$

Based on the bin set data:

$E_{bin}(1) > H(1)$; $E_{bin}(2) > H(2)$; and $E_{bin}(3) > H(3)$, which means that all three bins meet their threshold requirements. In normal operating mode, this would result in data from all three bins being stored in output memory. However, since there is a memory shortage, indicated by $pri\_fl(7) = 1$, and since the number of bins to store in a memory deprived situation is only one, indicated by $s_{pri} = 1$, then only data from the first bin are stored in final output memory.

As a result:

- Since three bins meet threshold requirements $anal\_qty(7) = 3$;
- Generated header information:

$$\mathbf{hdr}(7) = \left[7, anal\_qty(7), pri\_fl(7), \mathbf{anal\_ROI}(7) \right]$$
$$= [7, 3, 1, 1, 2, 3]$$

- Compressed FFT data output:

$\mathbf{X}_{out}(7) = [1, 2]$ and $\mathbf{Y}_{out}(7) = [0, 1]$;

- Therefore the final output is

$$\mathbf{Out}(7) = \left[\mathbf{hdr}(7), \mathbf{X}_{out}(7), \mathbf{Y}_{out}(7) \right]$$
$$= [7, 3, 1, 1, 2, 3, 1, 2, 0, 1] .$$

F. SUMMARY

This chapter provides a conceptual description of the primary design modules and data management strategies used to develop the SDR design. The FFT module processes the input IF signal so that it can be analyzed in the frequency domain. The Bin Energy Calculation module then associates the frequency domain data with the appropriate operator-defined bins. Once the data points for each bin are processed, the energy in each bin is calculated. The bin energies are processed by the Bin Threshold Analysis module, which passes analysis data onto the final Data Management module. The final module controls the process for reading FFT signal data from temporary memory, compresses the dataset, and stores the compressed dataset in final output memory. The next chapter focuses on the implementation of the conceptual model in the SysGen development environment. Detailed descriptions of each design element are provided in context with its parent module and any interrelated components.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. DESIGN IMPLEMENTATION DETAILS

Chapter III provided an overview of the SDR's conceptual design elements and the functional groupings of its signals and data values. This chapter focuses on the design implementation that was created using the SysGen software environment. Based on the resources available, the SDR was designed for a Virtex-4 FPGA architecture. Since hardware resources vary depending on the FPGA architecture, the use of a Virtex-4 affects the design's portability. However, a majority of the components used in the design are portable between Virtex-1 and Virtex-5 architectures. Figure 14 illustrates the high-level modules that constitute the full design. The modular descriptions provided in this chapter assume the reader has a basic familiarity with MathWorks' MATLAB and Simulink development tools.

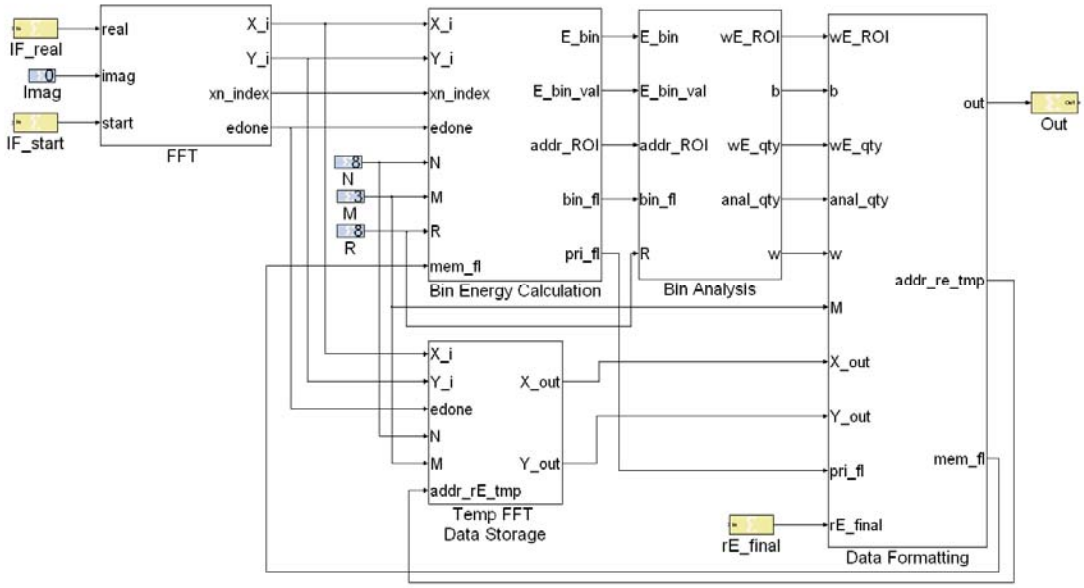


Figure 14. SysGen SDR Module.

Starting on the left side of the design, there a two SysGen Gateway In modules, IF_{real} and IF_{start}. These inputs are used to route sampled signal data from an external source into the design, directly to the FFT module. Since IF signals are always real, the FFT modules imaginary input is connected to a constant zero. As described in the last

chapter, FFT data are used to calculate energy in the Bin Energy Calculation module and they are also stored in the Temporary FFT Data Storage module for later compression. Bin energy calculations are then routed to the Bin Analysis module for comparison to established thresholds. Finally, analysis data are routed to the Data Formatting module. The analysis data are then used to manage the compression of FFT data that is stored in the Temporary FFT Data Storage module. The SysGen Gateway In module rE_{final} is used to control when the compressed data are read out of the design and sent out via the SysGen Gateway Out module *Out* .

As design elements are described in this chapter, three different types of elements will be discussed: SysGen IP cores (modules); Control algorithms, which are defined using MATLAB code and implemented via SysGen M-code modules; and control modules, which are interconnected compilations of the two.

A. FAST FOURIER TRANSFORM (FFT)

1. SysGen FFT v4.1 Module

The FFT in this design is implemented using the SysGen FFT v4_1 module shown in Figure 16. The input signals, IF_{real} and the constant zero are first passed through SysGen Delay modules. The z^{-4} displayed in each of the modules indicates that the input signals are delayed by four clock cycles before being applied to the FFT module's inputs, xn_re and xn_im .

$$xn\_re[n] = IF_{real}[n-4] \text{ , } n \geq 4 \qquad \text{(IV.1)}$$

$$xn\_imag[n] = 0 \qquad \text{(IV.2)}$$

As described in Eq. (II.3), the variable n represents the time index. The delay is included to compensate for the module's implementation which injects a four clock delay between the output index (*xn_index*) and the associated output data (*xk_re* and *xk_imag*). Figure 15 illustrates the timing relationship between each of the module's IO signals.

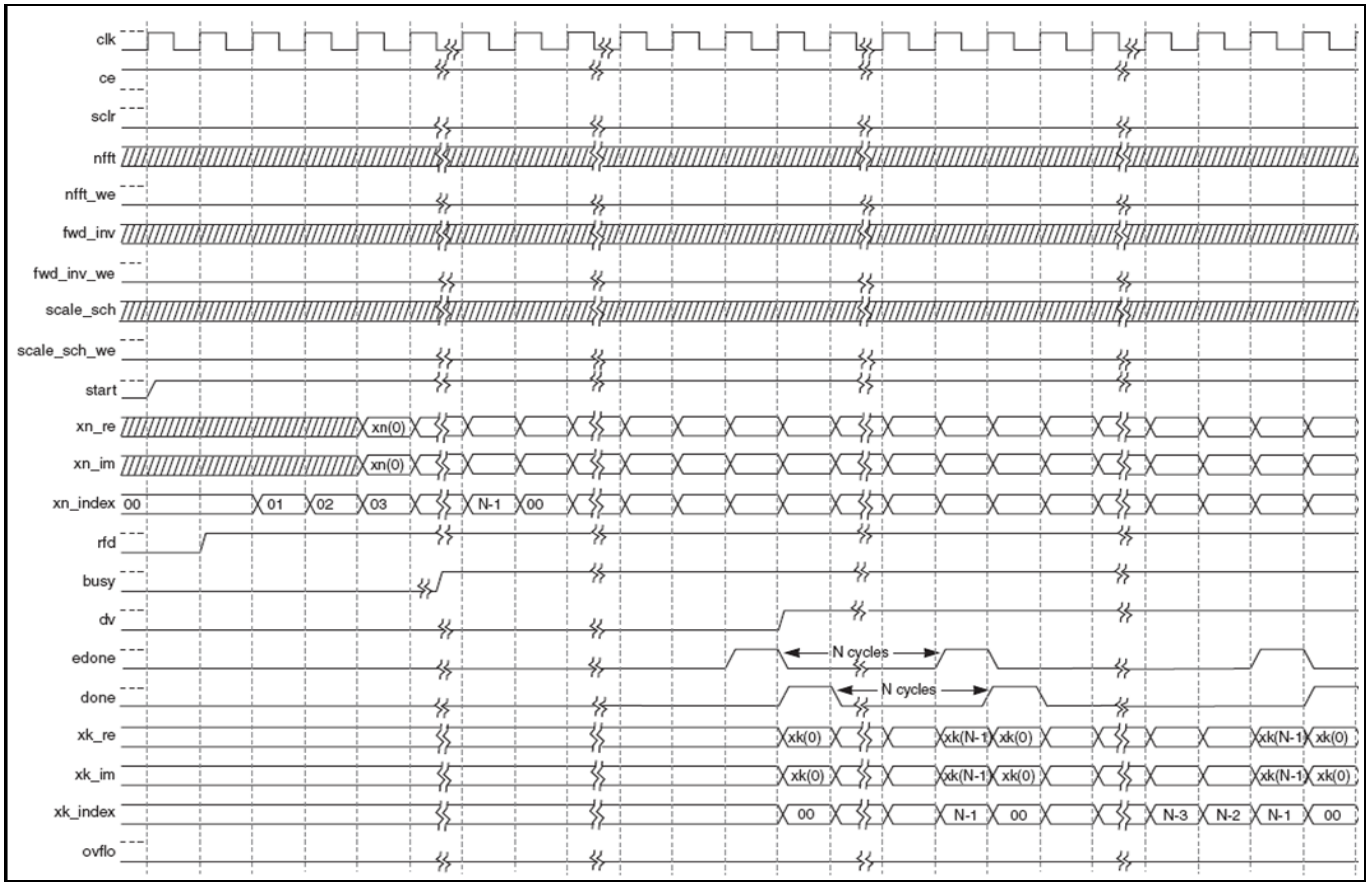


Figure 15. SysGen FFT v4.1 Module Timing Diagram.

As defined in Eq. (II.4), FFT data values are scaled by the number of points in the FFT, N. Although the SysGen FFT module is designed to enable the scaling process, the documented process was not clear during the design's development. To work around the issue the module was used without internal scaling, i.e., the scaling feature of the SysGen FFT module itself was disabled. The FFT module's output therefore was

$$G[k] = \sum_{n=0}^{N-1} g[n] e^{-j(2\pi/N)kn} \tag{IV.3}$$

This implementation differs from the definition in Eq. (II.4) by a scaling factor of $1/N$. Although in certain applications this could be a trivial matter, it could potentially lead to memory overflows in this design. As a result, the SysGen Scale modules illustrated in Figure 16 were utilized to translate the SysGen FFT module's output data to match the definition in Eq.(II.4). All resulting data points are stored in RAM for compression in a later stage of the design. The addressing scheme for this data will be covered in Section IV.D.3

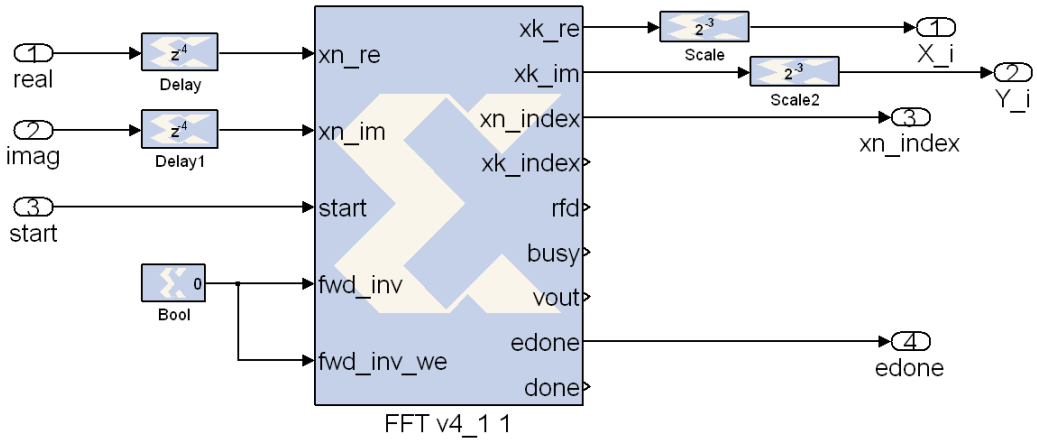


Figure 16. SysGen FFT Module (N = 8).

The module's relevant I/O signals are listed in Table 2 and Table 3. These signals are specific to the SysGen FFT module, but they interface directly with the signals defined in the previous chapter.

Table 2. Input Signals: SysGen FFT Module v4.1.

Signal	Functional Purpose / Description
xn_re	The real component of the input data stream. The driving signal can be a signed data type of width D with binary point at $D-1$, where D is a value between 8 and 24, inclusive (e.g.: Fix_8_7, Fix_24_23). $Fix\_D\_(D-1)$ indicates a D bit fixed-point numerical word with $D-1$ bits following the decimal point.
xn_im	The imaginary component of the input data stream. Same format as xn_re, however for this design the applied input is always zero.
start	Set to indicate *xn_re* and *xn_im* inputs are valid.
fwd_inv	Dictates whether the module processes input data as an FFT (forward) or as an IFFT (inverse). [0] = forward transform. [1] = inverse transform. The driving signal must be a Boolean type.

Table 3. Output Signals: SysGen FFT Module v4.1.

Signal	Functional Purpose / Description
xk_re	The real component of the output data stream.
xk_im	The imaginary component of the output data stream.
xn_index	Indicates the index of the input data that are being processed.
xk_index	Indicates the index of the output data. Note: There is a four clock delay between the output index and the associated output data; refer to timing diagram in Figure 15.
vout	A Boolean signal which indicates whether output data are valid or invalid.
edone	A Boolean signal that is active high one sample period before the block is ready to produce the processed data frame.

2. SysGen FFT v4.1 Module Constraints

As outlined in Table 2, the FFT module requires input data (*xn_re* and *xn_im*) to be formatted as a fixed point signed value in the form of $Fix\_D\_(D-1)$. Therefore, if D=8 then the input data format would be Fix_8_7. This formatting requirement dictates that signal input values applied to the FFT module must have magnitude less than one. This situation could place a major restriction on the magnitude of signals that are processed by the design. To compensate for the limitations, the IF_{real} input signal must be normalized by a constant (c). The actual SDR application will dictate the appropriate value for the constant, but for testing purposes c is set to 4.

$$xn\_re = IF_{real} / c \quad \text{(IV.4)}$$

$$xn\_imag = 0 \quad \text{(IV.5)}$$

The normalization defined in Eq. (IV.4) is accounted for in the Bin Threshold Analysis stage of the design.

Another potentially significant constraint of the SysGen FFT v4.1 module is that it is the only component in the SDR design that is not compatible with Virtex-1 hardware resources. It is portable to all other Virtex architectures, up to the Virtex-5. If this design were to be realized on a Virtex-1, this IP core would need to be replaced.

B. BIN ENERGY CALCULATION

As outlined in Chapter III.C, bin energy calculations require three basic steps: Calculate energy in each FFT index, use those values to calculate the energy in the time window, and then use the time window values to calculate the energy for each frequency window. As shown in Figure 17, signal data are routed from the FFT module to the windowing module, which consists of three energy calculation components. The Energy (FFT Index) module is responsible for calculating the energy in each FFT index, $E(k)$. These values are then passed on to the Energy (Time Window) module, which calculates energy spectrum for the time-window, $\mathbf{E}_{time}(w)$. That vector is then routed to the Energy (Frequency Window) module, which is responsible for determining the energy in each bin, $E_{bin}(b, w)$.

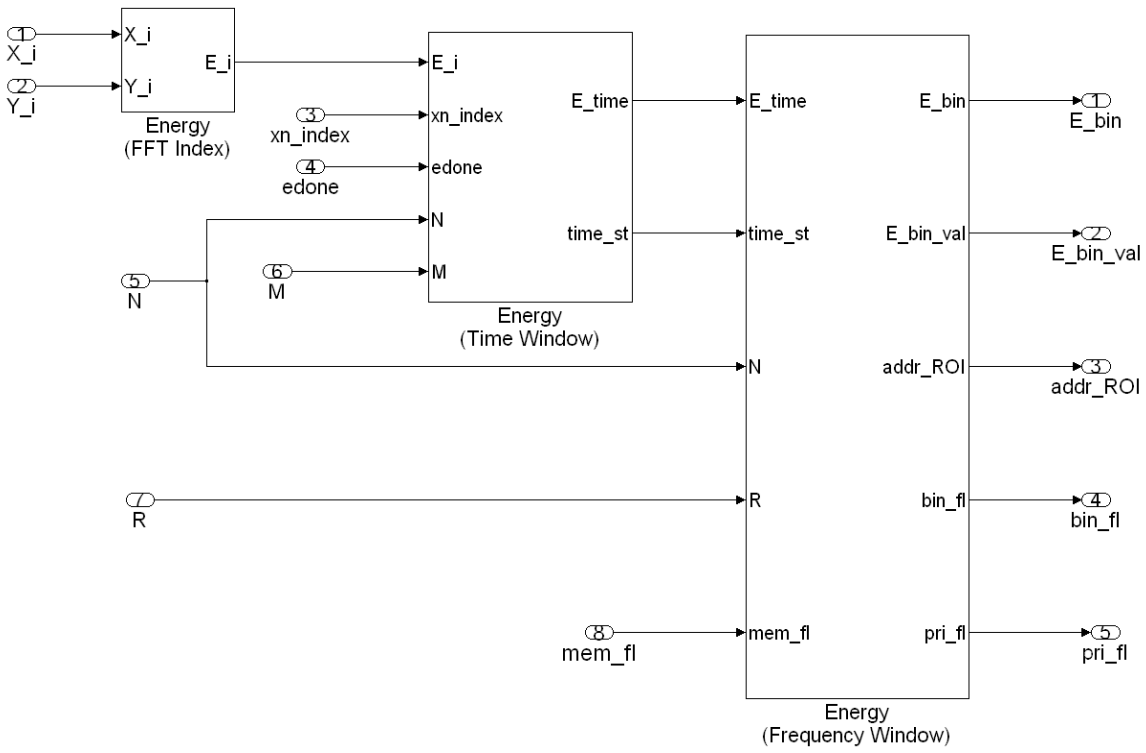


Figure 17. Control Module: Bin Energy Calculation.

1. Control Module: Energy (FFT Index)

As described by Eq. (III.5), the process for calculating energy at an FFT index involves adding the squared values of the real and imaginary components. This is implemented using two multipliers and an adder, as shown in Figure 18.

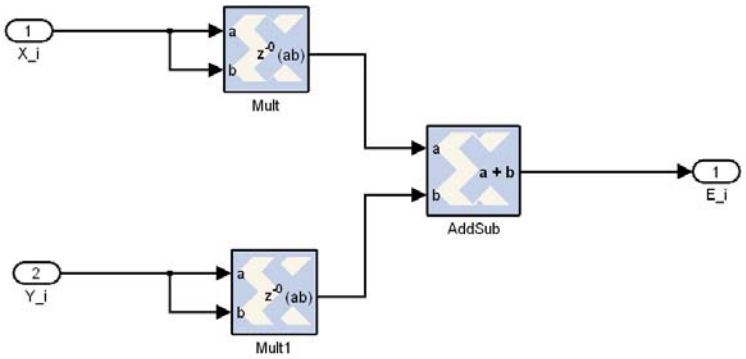


Figure 18. Control Module: Energy (FFT Index).

2. Control Module: Energy (Time Window)

In order to calculate the energy in a user defined time window, it is first necessary to track and store energy values associated with an FFT period, $\mathbf{E}_{min}(m)$, as described in Eq. (III.6). Then, on an FFT-index-based level, add the values for each vector associated with a time window. This is accomplished by working with first-in-first-out (FIFO) memory, an accumulator, multiplexers, and the *pwr_time* control algorithm that manages each element. The implementation can be viewed in Figure 19.

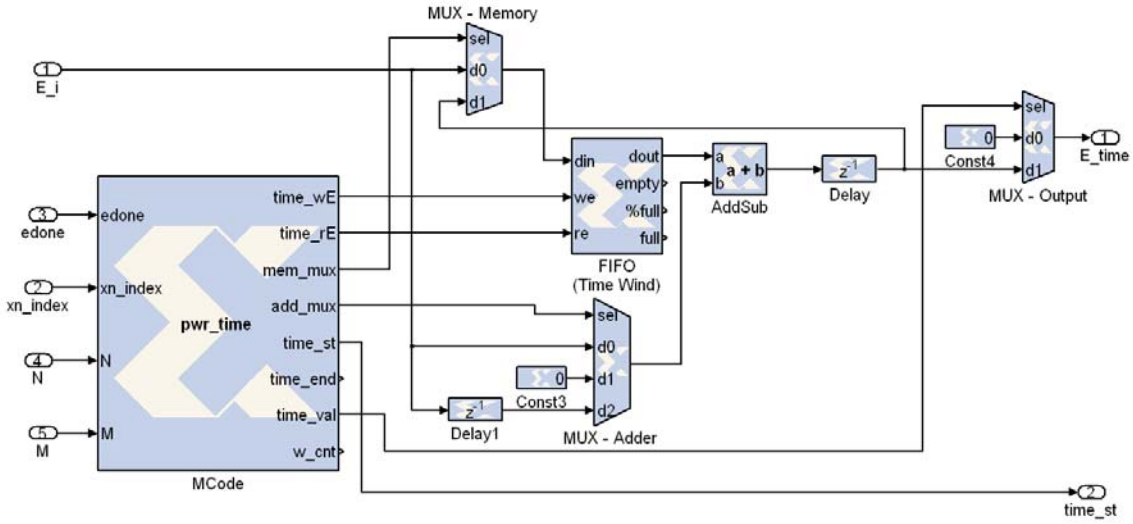


Figure 19. Control Module: Energy (Time Window).

As described in Chapter III.C, a key element to energy calculation in this design is knowing which data points associate with a time window vector $\mathbf{E}_{time}(w)$ and each of its $\mathbf{E}_{min}(m)$ vectors. To manage this information and the time windowing process, the *pwr_time* control algorithm was scripted using MATLAB code. The module utilizes the output ready flag, *edone*, and the discrete frequency signal, *xk_index*, signals from the SysGen FFT module to track data indices and control elements of the larger module. As described in, the *edone* signal indicates that data for a new FFT period will be generated on the next clock and *xk_index* indicates the FFT index, k.

The time windowing process involves storing the energy values of a bin set's first $\mathbf{E}_{min}(m)$ vector into FIFO memory. Then as $\mathbf{E}_{min}(m+1)$ values become available, they are added, by index, to the values stored in memory and the sums are stored in memory. In theory, this process would repeat for each FFT period in a time- window, which would be M times. However, in practice the *pwr_time* control algorithm only repeats the process M - 1 times and makes a slight revision on the last pass. The final vector, $\mathbf{E}_{min}(m+M-1)$ is added to the data in memory and the result is $\mathbf{E}_{time}(w)$. Instead of sending the final vector data to memory, it is routed to the next stage of the design. During this final stage, the control algorithm also generates signals to indicate when the energy values for a time-window start ($time\_st$) and when they stop ($time\_end$) being produced.

The *pwr_time* control algorithm generates several other control signals to manage all the components in the Energy (Time Window) control module. A complete list is provided in Table 4.

Table 4. Output Signals (*pwr_time*).

Signal	Functional Purpose / Description
$time\_wE$	Control when data are written to memory.
$time\_rE$	Control when data are read from memory.
add_mux	Control signal for MUX_Adder. Dictates which form of the data input stream is sent to the accumulation circuit.
mem_mux	Control signal for MUX_Memory. Dictates whether data written to memory is directly from $E_{ind}(i)$ or the Accumulator.
$time\_st$	Indicates the first output data point associated with a time-window.
$time\_end$	Indicates the last output data point associated with a time-window.
$time\_val$	Control signal for MUX_Output.

3. Control Module: Energy (Frequency Window)

As described in Chapter III.C.2, the process for calculating energy in a frequency window involves index-based analysis of the values in $\mathbf{E}_{time}(w)$, as illustrated in (III.14). To carry out this analysis, the vector data are first stored in random access memory. To manage how this information is stored, the Write Enable (Time Window) module was designed to control the input addressing scheme. Next, the stored energy values are read out of memory and summed in accordance with the bin indices stored in each of the $\mathbf{L}(b)$ vectors. The resulting values represent the energy in each bin, $E_{bin}(w,b)$. Two distinct control mechanisms were designed to facilitate this process. The first, Read Enable (Frequency Window), facilitates the process of reading data from memory. The second control module, Accumulator Control, is used to manage the accumulators that calculate $E_{bin}(w,b)$. The entire frequency windowing module can be seen in Figure 20, and more specific details are provided in the following sections.

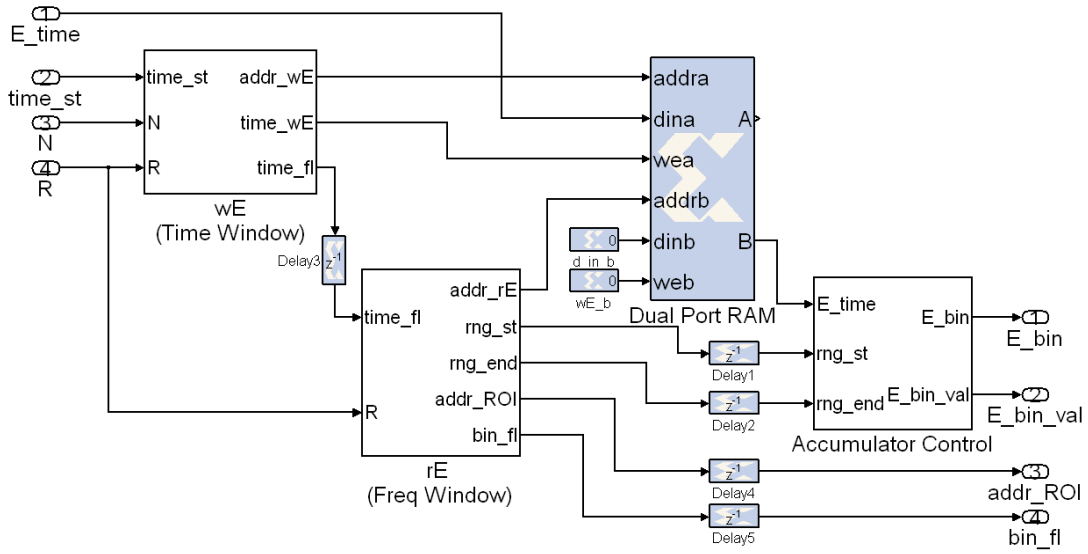


Figure 20. Control Module: Energy (Frequency Window).

a. SysGen Dual Port RAM Module

The SysGen Dual Port RAM module displayed in Figure 21 is used to store the time-window energy values, $\mathbf{E}_{time}(w)$.

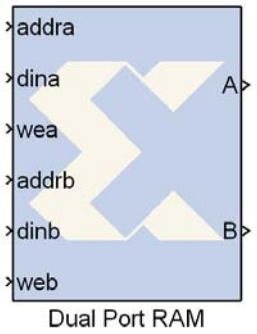


Figure 21. SysGen Dual Port RAM Module.

The benefit of the module is that it enables simultaneous access to two separate memory spaces, at different sample rates. This allows data to be written or read from one address (A), while data are read or written to a separate location (B). These functions are managed with control signals listed in Table 5.

Table 5. IO Signals: Dual Port RAM.

Signal	Functional Purpose / Description
addr _ a	Controls the address for memory location *A.*
din _ a	Data to be written to memory location *A.*
wE _ a	Write enable control for memory location *A.*
addr _ b	Controls the address for memory location *B.*
din _ b	Data to be written to memory location *B.*
wE _ b	Write enable control for memory location *B.*
A	Output for memory location *A* (1 clock delay).
B	Output for memory location *B* (1 clock delay).

b. Control Module: Write Enable (Time Window)

All $\mathbf{E}_{time}(w)$ data values are temporarily stored in RAM, so a fixed addressing scheme is required. For this SDR design, storage addresses are determined based on the time-window sequencing number, w, and each data value's FFT index, k. The two parameters are used to generate an address ($addr\_wE$), which is composed of two binary words. The most significant word ($addr\_wE_1$) is a function of w and the least significant word ($addr\_wE_2$) is a function of k.

$$addr\_wE_1 = w - 1 \tag{IV.6}$$

$$addr\_wE_2 = k \tag{IV.7}$$

$$addr\_wE = [addr\_wE_1, addr\_wE_2] \tag{IV.8}$$

c. Control Algorithm: wE_time_win

The *wE_time_win* control algorithm was designed, using MATLAB code, to implement the addressing scheme described in Equation (IV.8). When the algorithm receives the control signal that indicates the start a new $\mathbf{E}_{time}(w)$ vector, $time\_st$, it generates a write enable signal ($time\_wE$) to control the RAM's input storage mechanism. The algorithm works with control signals N and R to manage the addressing scheme. As a reminder, N is a constant that indicates the number of samples per FFT period. The variable R is a constant that indicates the maximum number of bin sets that can be stored in memory before the bin index, b, starts over at zero. The algorithm is designed to produce the two address components starting with $addr\_wE_1 = 0$ and $addr\_wE_2 = 0$. At each clock cycle, the module increments $addr\_wE_2$ until it equals $N-1$. For that entire period, $time\_wE$, remains active high so that storage RAM is able to store generated data. At the end of the period, the algorithm sets $time\_wE$ to zero, it sets $time\_fl$ equal to one, and then awaits the next $time\_st$ flag. The $time\_fl$ flag indicates that the last $\mathbf{E}_{time}(w)$ data value has been saved into memory. When $time\_st$ is reset to one, the algorithm updates the address

variables so that $addr\_wE_1 = 1$, $addr\_wE_2 = 0$, and the steps above repeat. This process is repeated until $addr\_wE_1$ is equal to the number of bin sets that can be stored in memory, R, and then the address resets to the initial state where $addr\_wE_1 = 0$ and $addr\_wE_2 = 0$.

The final element of the addressing scheme requires a SysGen concatenation module. This block concatenates $addr\_wE_1$ with $addr\_wE_2$ to generate the actual address $addr\_wE$. The full implementation can be seen in Figure 22.

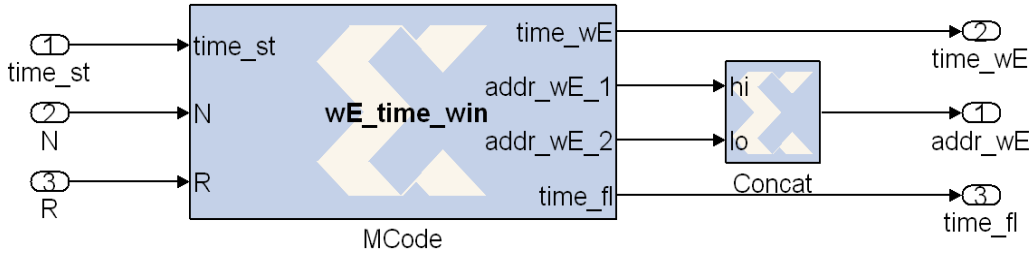


Figure 22. Control Module: Write Enable (Time Window).

d. Control Module: Read Enable (Frequency Window)

Once all the $\mathbf{E}_{time}(w)$ data are stored in memory, a control mechanism is required for reading out the values associated with operator-defined ranges. As shown in Figure 23, this design utilizes two RAM modules for *ROI* Control memory elements, the rE_freq_win control algorithm, the *ROI* Quantity control module, and a concatenation block to manage the process.

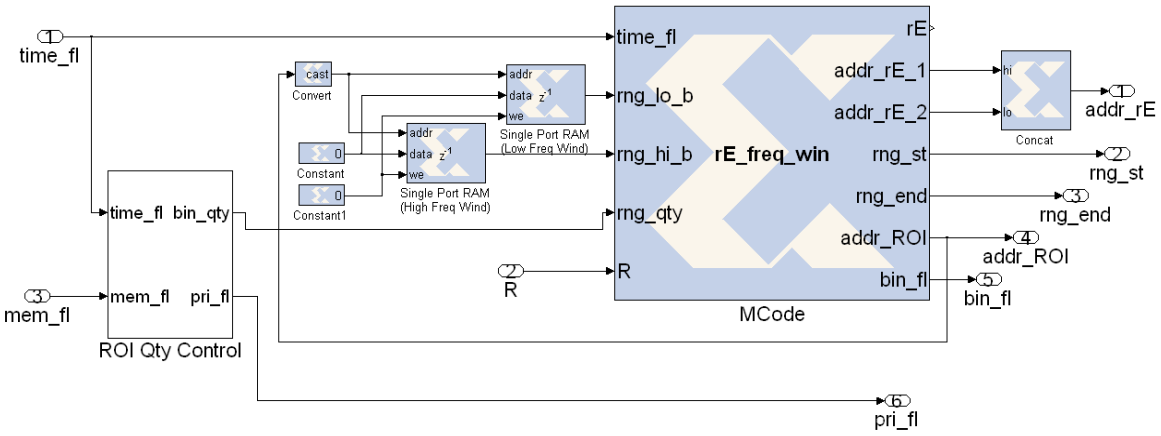


Figure 23. Control Module: Read Enable (Frequency Window).

In this design, *ROI* Control memory consists of two SysGen Single Port read-only-memory (RAM) modules, as illustrated in Figure 23. *Single Port RAM (Low Freq Wind)*, contains the starting indices for all the *ROIs*, while *Single Port RAM (High Freq Wind)* contains the final indices for each. The most important point about the memory elements is the way in which range information is stored. The memory address ($addr\_ROI$) for each *ROI* is a function of its associated bin, b, and is the same for both RAM modules,

$$addr\_ROI(b) = b - 1 \qquad \text{(IV.9)}$$

This means, for example, that the starting index for the 3rd *ROI* is stored in *Single Port RAM (Low Freq Wind)* at $addr\_ROI(3) = 2$ and the final index for the *ROI* is stored in *Single Port RAM (High Freq Wind)* at the same address.

e. Control Module: ROI Quantity Control

The *ROI* Quantity control module performs two simple functions. First, it is responsible for establishing the design's operational mode during a bin set's analysis period. As mentioned in Section III.E, the design relies upon the $pri\_fl(w)$ control signal to determine the appropriate mode. As described in the next section, the signal's status determines the number of bins analyzed during each bin set, bin_qty. As illustrated in Figure 24, this is accomplished using the *mem_pri* control algorithm, a

MUX, and the operator-defined constants, s and s_{alt}. The variable s indicates the default number of bins analyzed when the system's output memory capacity is below its threshold, U. During periods when this is not the case, s_{alt} indicates the alternate number of bins to be processed.

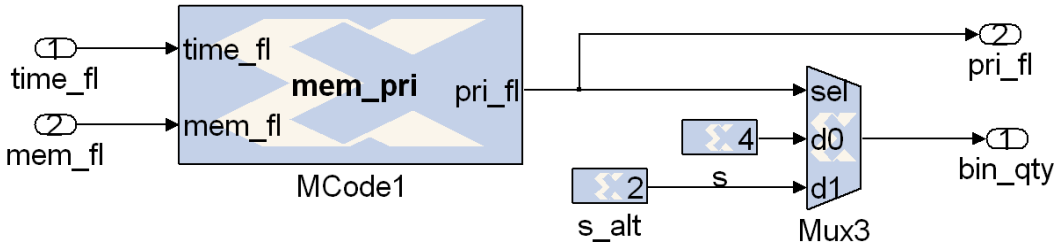


Figure 24. Control Module: ROI Quantity Control.

f. Control Algorithm: mem_pri

The *mem_pri* control algorithm's sole purpose is to set the $pri\_fl(w)$ flag which indicates the status of the design's operating mode during each bin set's analysis period. The $pri\_fl(w)$ flag is a function of the flag that indicates whether final output memory has exceeded the operator-defined threshold capacity, mem_fl. If the mem_fl is zero when the bin set's analysis period begins, indicated when $time\_fl$ is set to one, then pri_fl is reset to zero until the next $time\_fl$ pulse is received. However, if mem_fl is one when $time\_fl$ is pulsed active-high, then pri_fl is set to one until the next $time\_fl$ pulse is received.

g. Control Algorithm: rE_freq_win

One of the most important functions of the Read Enable (Frequency Window) control module is to generate signals that facilitate management of each bin's frequency windowing process. The rE_freq_win control algorithm was designed using MATLAB code to orchestrate the process. When the algorithm receives the $time\_end$ flag, which indicates that the last energy value of a time-window has been

calculated, it sets two control signals: (1) The rng_st signal is pulsed active-high to indicate that the first address of a bins stored time-window data, $\mathbf{E}_{time}(w)$, has been generated, and (2) the signal $addr\_ROI(b)$ signal is set as described in Eq. (IV.9). The second signal is used to control the output of the ROI Control memory, which provides start (rng_lo) and stop (rng_hi) indices for each ROI. The algorithm also sequentially generates the address values ($addr\_rE$) for memory locations associated with each ROI. Since this process is affected by the addressing used in the wE_time_win control algorithm, the rE_freq_win algorithm utilizes a similar two word addressing algorithm where the most significant word ($addr\_rE_1$) is a function of w.

$$addr\_rE_1(w) = w - 1 \tag{IV.10}$$

The difference in the rE_freq_win algorithm is that the least significant word ($addr\_rE_2$) does not necessarily start at zero and end at N -1. Instead, the set of indices for each ROI ($\mathbf{addr\_rE}_2$) begins with the relevant rng_lo value and ends with the appropriate rng_hi value.

$$\mathbf{addr\_rE}_2(b) = [rng\_lo(b),\ rng\_lo(b)+1,\ \ldots,\ rng\_hi(b)] \tag{IV.11}$$

The algorithm generates each element of $\mathbf{addr\_rE}_2$ sequentially. The individual values represent the second word of the address, $addr\_rE_2$.

$$addr\_rE(w,b) = [addr\_rE_1, addr\_rE_2] \tag{IV.12}$$

When the last address of the initial bin is generated, the rE_freq_win control algorithm again generates two control signals. However instead of rng_st, rng_end is pulsed active high to indicate that the last address for the bin has been generated; $addr\_ROI$ is then set to one in order to force ROI Control memory to provide rng_lo and rng_hi indices for the next ROI. At the next clock pulse, rng_st is once again pulsed active-high to indicate the beginning of addresses for a new bin. As described in Section IV.B.3.e, the process for computing $addr\_set(b)$ for the second range is the same as described above and the procedure is repeated for the number of bins

designated by the operating mode, bin_qty. When the final address for a bin set is generated, the re_freq_win control algorithm pulses the bin_fl control signal active-high as an indicator.

Three additional points should be highlighted about the algorithm. First, since ROI Control memory can be preconfigured to hold several unused $ROIs$, it is important to set the appropriate default number of bins to be analyzed per bin set, s. It is also critical to ensure all ROI data are stored in priority order. Finally, the number of bins analyzed in a bin set, bin_qty, is not necessarily the same during all stages of the design's operation. If the flag that controls the design's operating mode, $pri\_fl(w)$, is set to zero then bin_qty is equals the default value, s. Otherwise bin_qty is equal to the alternate number of bins per bin set, s_{pri}.

h. Control Module: Accumulator Control (Frequency Window)

As the time-window energy spectrum data, $\mathbf{E}_{time}(w)$, are read out of memory, it is necessary to manage how values in each ROI are summed together. The process is facilitated using the $accum\_ctrl$ control algorithm, two accumulators, and a multiplexer, as shown in Figure 25.

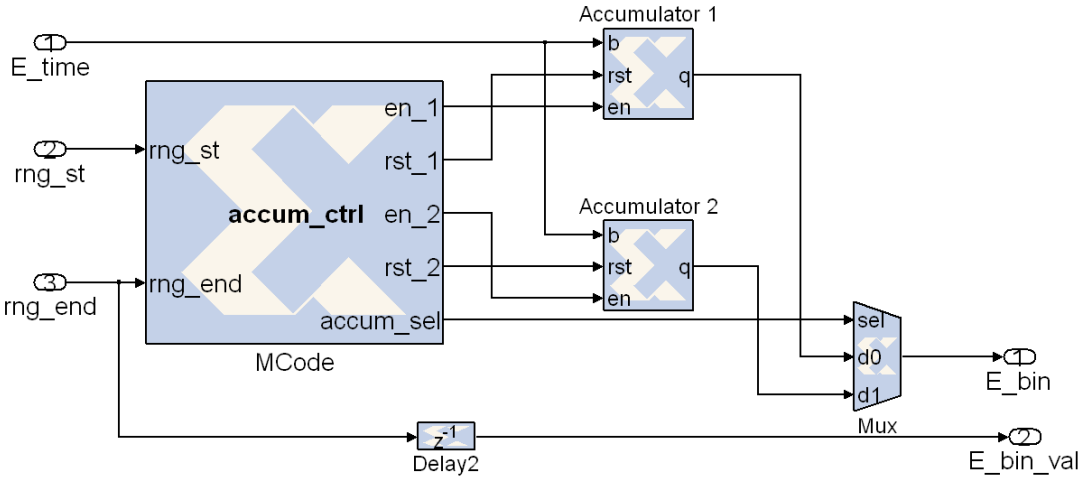


Figure 25. Control Module: Accumulator Control.

i. Control Algorithm: accum_ctrl

The $accum\_ctrl$ control algorithm was designed using M-code to manage the accumulators and MUX that are used to calculate the total energy in each bin, $E_{bin}(w,b)$. When the algorithm receives the flag which indicates the first address of a ROI has been generated, rng_st, it enables one of the two accumulators by setting en_1 or en_2 equal to 1. For this description, assume *Accumulator_1* is chosen to start, so en_1 is set active-high. This choice also dictates that the output MUX control signal, $accum\_sel$, is set to zero so that the final $E_{bin}(w,b)$ value comes from the appropriate accumulator. The $accum\_ctrl$ algorithm continues to assert en_1 until it receives the flag which indicates that the last address of the ROI has been generated, rng_end. During that interval, *Accumulator_1* sequentially adds $\mathbf{E}_{time}(w)$ values and calculates the total energy, as described in Eq. (III.7). When the $ROI's$ energy calculation is complete, the module pulses the E_{bin}_val signal active-high as an indicator. There is a one-cycle delay before calculated sums are available at the accumulator's output, and this impacts when the device can be reset and when the MUX control signal can be updated. Therefore, two clock cycles after receiving rng_end, the $accum\_sel$ signal is changed from zero to one and the signal that resets the accumulator, rst_1, is pulsed high. Despite these inherent delays, if a new rng_st signal were received one clock after a rng_end signal, the control algorithm is designed to enable $Accumulator\_2$ for immediate processing of the next ROI. The major point is that the $accum\_ctrl$ algorithm is designed to switch accumulators for each new ROI.

C. BIN THRESHOLD ANALYSIS

Data generated in the Bin Energy Calculation control module provides information about the energy in each operator-defined range of interest. The Bin Threshold Analysis control module is designed to analyze this information and store relevant parameters for additional processing. The major components of the module, as

illustrated in Figure 26 are a RAM module to store each bins threshold value (H), a comparator module, delay elements, and the *bin_analysis* control algorithm.

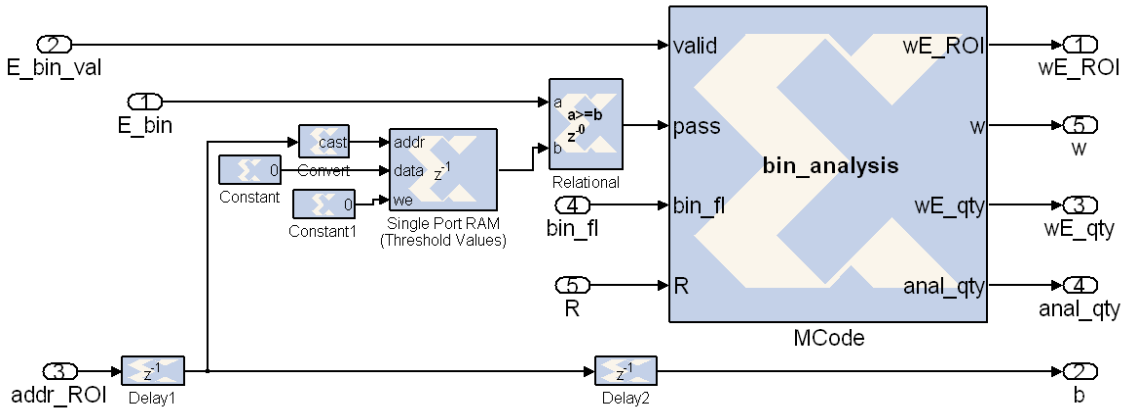


Figure 26. Control Module: Bin Threshold Analysis.

The initial function of the module is to compare all calculated bin energy values, $E_{bin}(w,b)$, to their operator-defined thresholds, $H(b)$. To do this, threshold values are stored in memory locations, $addr\_thresh(b)$, that correspond with their associated bin index, b.

$$addr\_thresh(b) = b - 1 \qquad \text{(IV.13)}$$

As the $E_{bin}(w,b)$ data values are generated, the ROI memory address signal introduced in Section IV.B.3.d, $addr\_ROI(b)$, is used to generate the appropriate $addr\_thresh(b)$ values. This enables the design to send energy and threshold values to the comparator for analysis. If a bin energy meets or exceeds its threshold, $H(b)$, the comparator sets the $pass$ signal active-high for one clock. Based on the results, the Bin Threshold Analysis control module generates analysis data that will be described in the following sections.

The $bin\_analysis$ control algorithm is designed using MATLAB code to capture three basic parameters about each bin: the total number of bins that meet threshold requirements, $anal\_qty$; the associated bins, b; and the time-window sequence value, w. The algorithm is built around three control signals: $pass$, which is generated by the

comparator; E_{bin}_val, which indicates that a bin's energy has been calculated; and bin_fl from the re_freq_win algorithm, which indicates that the last address for a bin set has been generated.

The algorithm essentially idles until the E_{bin}_val signal is set to one. The design then utilizes sequential $pass$ signals to calculate the number of bins in the bin set that meet threshold specifications, ($anal\_qty$). The $anal\_qty$ value is incremented every time the $pass$ signal is asserted until the bin_fl signal is received, which indicates that the last bin energy calculation has been completed. Upon receipt of this flag, if $anal\_qty$ is greater zero the module generates a write control signal (wE_qty) which forces the $anal\_qty$ value and the time-window sequence value, w, into memory. During the analysis phase, the $pass$ signal is also used to manage which bin indices are stored. Each time a bin's calculated energy meets its designated threshold, $H(b)$, the algorithm generates a separate write control signal (wE_ROI) to ensure its index, b, is captured in memory.

D. DATA MANAGEMENT

The final stage of this SDR design utilizes analysis data from the Bin Threshold Analysis control module to determine which FFT data points are read from temporary memory and then stores them in a final compressed format. Figure 27 shows the three principle control modules used to facilitate the process: Header Generator, Temp Data Control, and Output Format. Although not shown, the Temporary FFT Data Storage module is another integral component that will be discussed in this Data Management section.

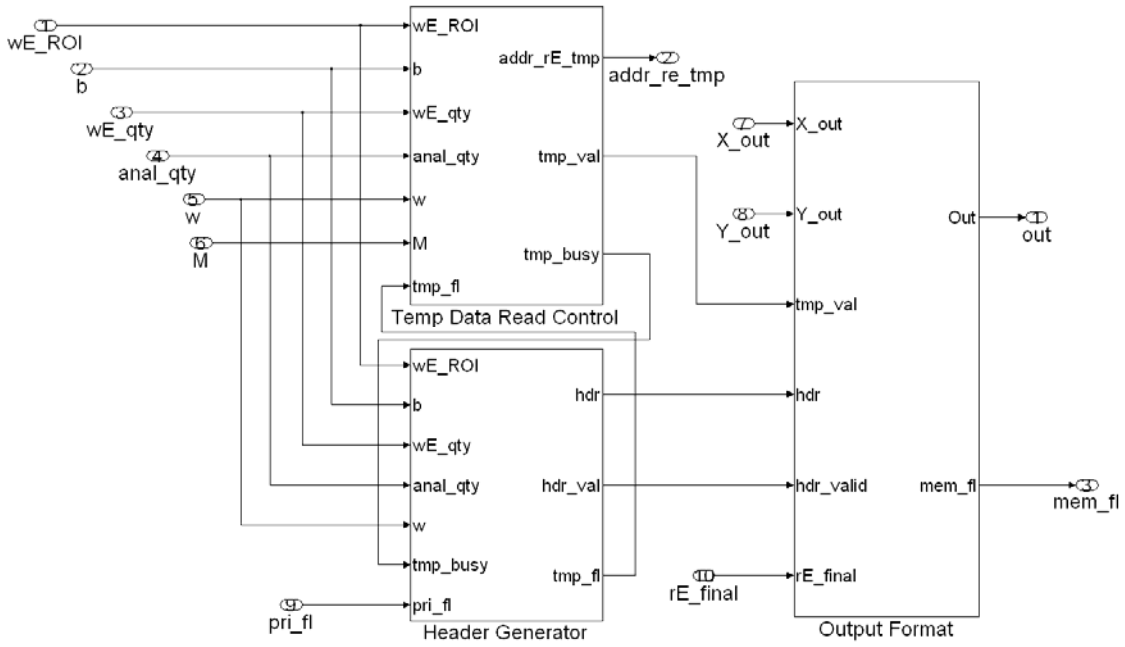


Figure 27. Control Module: Data Formatting.

1. Control Module: Temporary FFT Data Storage

As mentioned in Section III.B, every FFT data point is stored in temporary memory until its bin analysis is completed. The Temporary FFT Data Storage control module uses the *wE _ temp _ fft* control algorithm, a concatenation block, and two dual-port RAM modules to manage the addressing scheme. The physical connections between the modules can be seen in Figure 28.

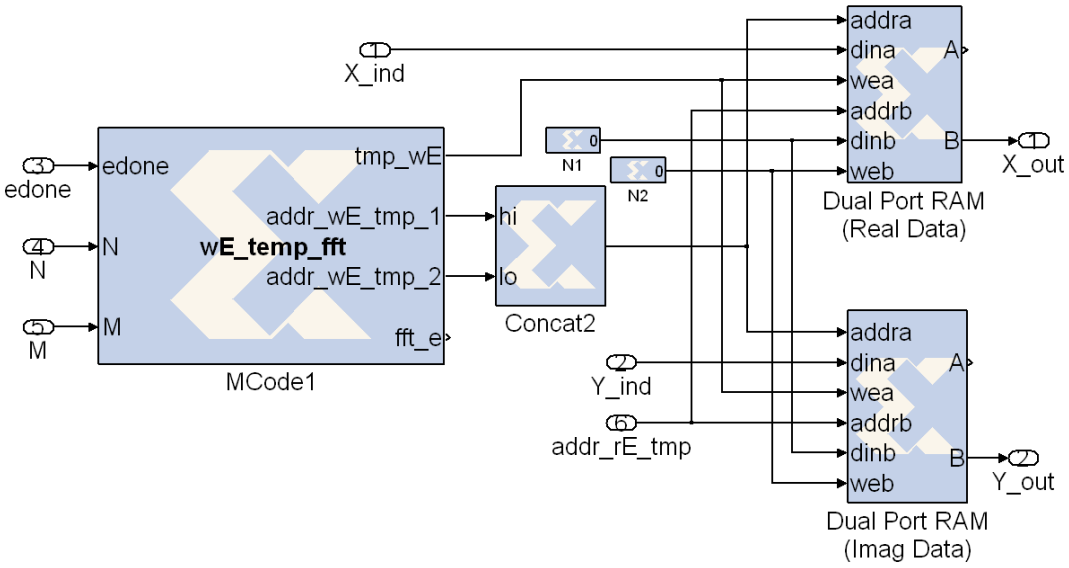


Figure 28. Control Module: Temporary FFT Data Storage.

The wE_temp_fft control algorithm is designed using MATLAB-code to ensure every valid FFT signal is stored in an indexed memory address ($addr\_wE\_tmp$). The indexing scheme is based on each signal's time-window sequence value, w, and the FFT index, k. These parameters are associated with two binary words that are concatenated to generate the final address. The most significant word ($addr\_wE\_tmp_1$) is a function of the signal's FFT sequence number (v),

$$addr\_wE\_tmp_1 = v - 1 \tag{IV.14}$$

It is important to note that v is different than the index m, which represents the continuously incrementing FFT period number. The FFT sequence numbers begin at one and increment to a maximum value (v_{max}) before starting again at one. v_{max} is a function of the number of FFT periods per time-window, M, and the constant used to control address rollover, R.

$$v_{max} = MR - 1 \tag{IV.15}$$

The index scheme's second word ($addr\_wE\_tmp_2$), relays information about each signal's position in an N-point FFT.

$$addr\_wE\_tmp_2 = k - 1 \tag{IV.16}$$

The values for $addr\_wE\_tmp_2$ begin at zero and are incremented to a maximum value of $N-1$. The two address components generated by wE_temp_fft are routed to a SysGen concatenation block, which generates the memory location for each FFT signal

$$addr\_wE\_tmp(v,i) = [addr\_wE\_tmp_1, addr\_wE\_tmp_2]. \qquad \text{(IV.17)}$$

Every time a valid address is generated, the algorithm sets the tmp_wE flag equal to one. Notice in Figure 28 that the same address is used for two separate memory blocks, one for real FFT values and the other for the imaginary components.

2. Control Module: Header Generation

When the $bin\_analysis$ control algorithm generates a wE_qty signal, it is an indication that all $ROIs$ in a bin set have been analyzed and the system has stored the data required to control the compression algorithm. As discussed in Section III.E, the compressed data set requires a header. The Header Generation control module generates headers using three FIFO memory blocks, two delay elements, and the two control algorithms (hdr_st_mgr and hdr_out), shown in Figure 29.

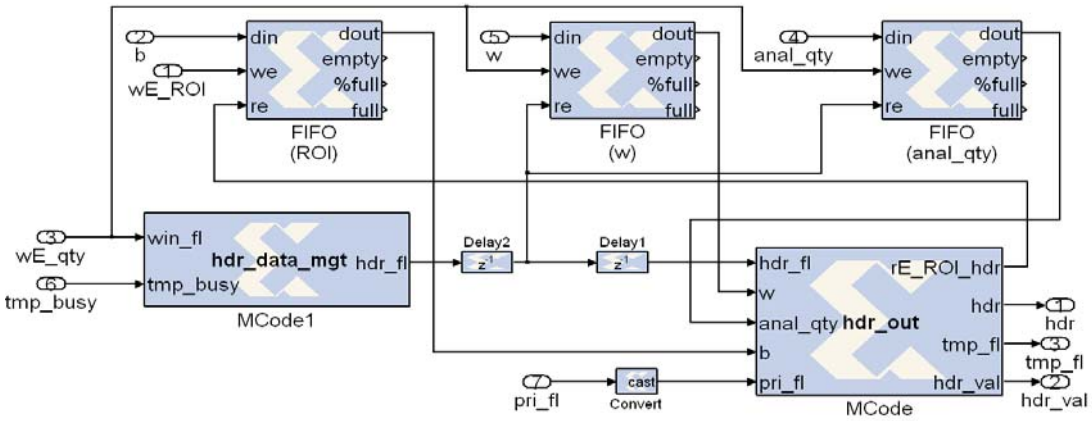


Figure 29. Control Module: Header Generation.

a. Control Algorithm: hdr_st_mgr

Whenever the design has data being written to final output memory, the tmp_busy control flag is set equal to one as an indicator. Although the wE_qty signal indicates that a bin set is ready for compression and storage, if the Header Generation control module started generating a new header while tmp_busy and wE_qty signals are both one, a data collision would occur at the final output memory. To prevent this from happening the hdr_st_mgr control algorithm was designed using MATLAB code to monitor both signals and generate a control flag, hdr_fl, which initiates the header generation for a newly processed bin set. Based on the cases outlined in Table 6, the control algorithm generates the appropriate value for the hdr_fl flag.

Table 6. Control Algorithm: *hdr_st_mgr*.

wE_qty	tmp_busy	hdr_fl Response
0	0	0
0	1	0
1	0	Pulsed high immediately.
1	1	Pulsed high as soon as tmp_busy returns to zero.

b. Control Algorithm: hdr_out

When the hdr_fl is set equal to one, the system is ready to start generating a new header for the most recently processed bin set. The hdr_out control algorithm was designed using MATLAB code to read analysis data from the appropriate storage modules and generate the header, $\mathbf{hdr}(w)$, as described in Eq. (III.19). When the algorithm receives the hdr_fl signal, it also receives the time-window sequence number, w, which is the first data element appended to the header. During that same clock cycle, the module receives the control signal that represents the number of bins that

met threshold requirements for the bin set, $anal\_qty(w)$. This datum is the second element appended to the header and it also controls the number of qualifying bin indices read out of the analysis memory. To manage the process, the algorithm generates a read enable signal (rE_ROI_hdr) for $anal\_qty(w)$ sequential clocks. This ensures the appropriate data are read out of memory and appended to the header's fourth segment, which represents the bin indices that met threshold requirements, $\mathbf{anal\_ROI}(w)$. The third element of the header is the flag that dictates the design's operating mode for the bin set analysis, $pri\_fl(w)$. When the last header bit is generated, the hdr_out control algorithm indicates the fact by setting the tmp_fl control signal active-high. During the entire period that **hdr**(w) data are being generated, the algorithm sets the hdr_val control signal equal to one as an indicator. The signal is used in the final stage to control the data output format.

3. Control Module: Temporary Data Read Control

With the header fully generated, the design is primed to read a subset of FFT data from temporary memory so it can then be stored in final output memory. To control the reading process the Temporary Data Read Control module, shown in Figure 30, is designed using three FIFO memory elements, two RAM modules for *ROI* Control memory, a concatenation block, and the rE_temp control algorithm.

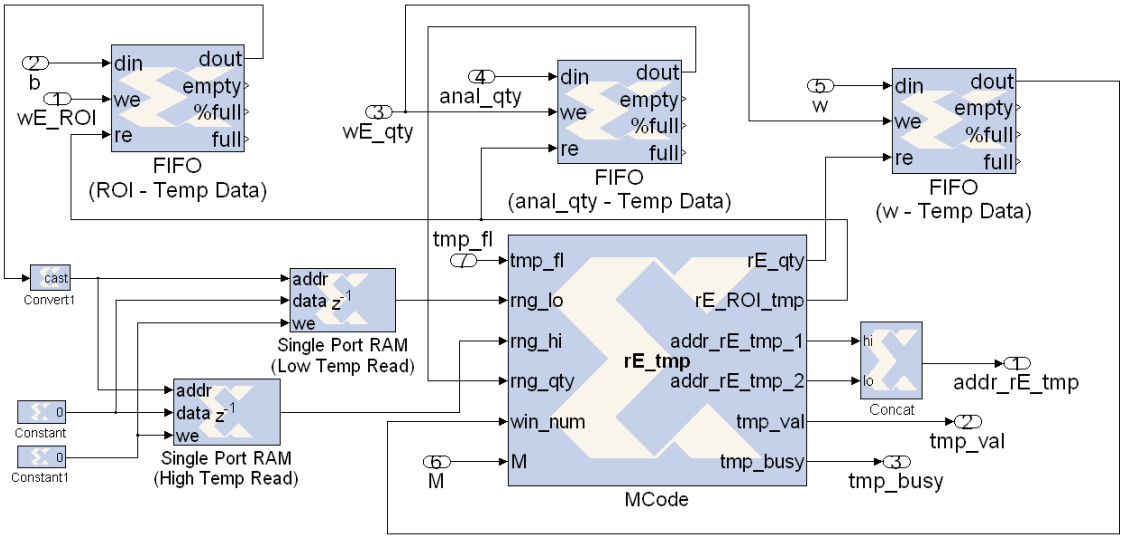


Figure 30. Control Module: Temp Data Read Control.

The rE_temp algorithm is designed using MATLAB code to evaluate analysis data for the purpose of selecting which FFT data to read out of temporary memory. When the algorithm receives the flag that indicates the last header bit has been generated, tmp_fl, it sets two distinct read enable signals to one, rE_qty and rE_ROI_tmp. The algorithm sets rE_qty high once per bin set, in order to control when the number of bins that met threshold requirements for the bin set, $anal\_qty(w)$ and the time-window index, w, values are read from memory. The rE_ROI_tmp signal controls when each qualifying bin index, b, is read from memory. It is asserted a total of $anal\_qty(w)$ times per bin set.

Based on analysis data, the rE_temp control algorithm generates the addresses ($addr\_rE\_tmp$) from which FFT data are read out of Temporary FFT Data Storage. Similar to wE_temp_fft, the algorithm's addressing scheme is based upon two binary words. The most significant word, $addr\_rE\_tmp_1$, is defined as a function of the time-window sequence number, w, the number of FFT periods per time-window, M, and the FFT sequence number, v.

$$addr\_rE\_tmp_1 = wM + v \tag{IV.18}$$

The least significant word, $addr\_rE\_tmp_2$, is a function of the desired position in the $N-$point FFT. The full address is a concatenation of the two binary words

$$addr\_rE\_tmp = [addr\_rE\_tmp_1, addr\_rE\_tmp_2] \quad \text{(IV.19)}$$

During the entire period that $addr\_rE\_tmp$ data are being generated, the algorithm sets the tmp_val control signal equal to one as an indicator. This signal is used in the final Output Format control module to facilitate the data formatting process.

4. Control Module: Output Format

The final stage of this SDR design is the Output Format control module. The module is responsible for packaging the header and the compressed FFT data and then routing them to final output memory. This last stage also generates the mem_fl flag, which indicates the status of available output memory. The components required to design the module are shown in Figure 31 and include a concatenation block, a multiplexer, an inverter, a FIFO memory element, a logical comparator, and a relational comparator.

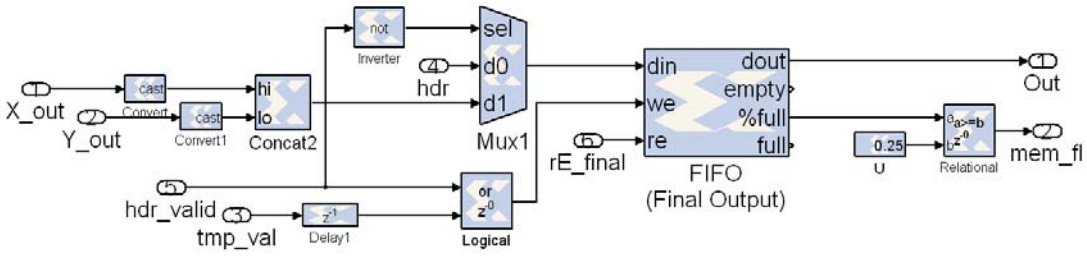


Figure 31. Control Module: Output Format.

The primary purpose of the module is to route and store bin set headers, $\mathbf{hdr}(w)$, and compressed FFT data, $\mathbf{X}_{out}(w)$ and $\mathbf{Y}_{out}(w)$, to the same final output memory element. Both data types are routed through a single multiplexer so the control signal that indicates valid header data, hdr_valid, is used to determine which type is passed to the MUX output. It is important to note that before FFT data are routed to the MUX, the real and imaginary components are concatenated. The real component is used as the most

significant word so that final data format is as described in Eq. (III.20). As data passes through the MUX to final memory the hdr_valid and a delayed tmp_val signals serve as write enable controls. The tmp_val signal is generated in the *rE_tmp* control algorithm to indicate that a valid temporary storage address has been generated. This algorithm uses a delayed version of the signal to compensate for the clock delay between the address generation and subsequent data output that is read from the Temporary FFT Data Storage control module. In order to read stored data out of this final stage of the design, an external source must provide an active high signal to the rE_final input node shown in Figure 31.

The Output Format control module is also designed to generate the mem_fl flag, which provides an indication of available memory capacity. The SysGen FIFO module has a $\%full$ output port, which indicates the percentage of memory in use at any given time. This signal is routed to the comparator and evaluated against an operator-defined memory threshold (U). If the percentage of memory in use is greater than this threshold, mem_fl is set to one, otherwise it is zero.

$$mem\_fl = \begin{cases} 1, & \%full > U \\ 0, & \%full < U \end{cases} \tag{IV.20}$$

E. SUMMARY

This chapter provided specific details as to how the SDR conceptual design model was implemented using SysGen software. Each design element is described in terms of its functional purpose and operational mechanics. Three different categories of design elements are described: SysGen IP cores (modules); control algorithms, which are defined using MATLAB code; and control modules, which are interconnected compilations of the two. As described in Chapter III, the primary control modules are the FFT, Bin Energy Calculation, Bin Threshold Analysis, and Data Management modules. Each primary control module was built using a combination of SysGen modules, control algorithms and sub-control modules. The following chapter explains the tests used to validate the designed SDR's functional operation. The chapter also provides test results and analysis.

THIS PAGE INTENTIONALLY LEFT BLANK

V. DESIGN TESTING

In order to validate the SDR's operational capability, it was necessary to verify the scaling requirements and its functional mechanisms. To evaluate the scaling properties, the design was built and tested using two separate versions. The first utilized an 8-point FFT and the second was built with a 1024-point FFT. The functional results of the two versions are captured throughout this chapter. There were three essential functional operations that required testing:

1. If an IF signal containing a single frequency is applied to the SDR's input for varying durations in a time-window, can the system properly calculate the associated energy and then generate the appropriate compressed output?
2. If an IF signal containing multiple frequencies is applied to the SDR's input for varying durations in a time-window, can the system properly calculate the energy in the different frequency bands and then generate the appropriate compressed output?
3. If available output memory drops below the threshold of 1-U, can the system adjust its output so that a maximum of s_{pri} bins are included in the **Out**(w) data vector?

The test procedures developed for the three functional areas are described in the following sections. All the tests were conducted using the SysGen SDR model and digitally generated signals. The design's outputs were captured electronically and then processed with a MATLAB file (M-file). The M-file decompressed the output data and generated the plots that are included below.

A. SINGLE FREQUENCY INPUT TEST

In order to verify the SDR's response to single frequency input signals, a sinusoidal signal, $IF[n]$, was applied to the system input for varying durations of the test cycle. The test cycle was defined as three time-window periods, $3T$, and the time-

window, T, was defined as three FFT periods, $M = 3$. To simplify analysis of the test results the period of the input signal was set equal to one FFT period. During the first time-window, the sinusoid was applied to the system input for one FFT period. During the second time-window, the signal was applied for $2T_{min}$. During the third and final time-window, the signal was applied for $3T_{min}$. To test the design, a bin was defined such that its ROI was equal to the digital frequency of the input signal. The bin's energy threshold, $H(1)$, was set equal to the energy contained in two T_{min} periods, of the sinusoid, $H(1) = 0.0312$. Based on these established bin parameters and the applied input signal, the SDR should only calculate a bin energy equal to or above the threshold for the last two time-windows. As a result, the system's output should only contain FFT data for the last two time-windows. After decompressing the output data stream and evaluating its inverse Fast Fourier Transform (IFFT), the resulting waveform, $\mathbf{C}(k)$, should be identical to the input except for during the first time-window. To clarify, decompression involves inserting zeros into all the FFT indices not included in the bin set's output, $\mathbf{Out}(w)$, and storing the new data set in the vector $\mathbf{D}(k)$. Therefore, $\mathbf{C}(k)$ is the IFFT of $\mathbf{D}(k)$

$$\mathbf{C}(k) = \text{IFFT}\left[\text{D}(\text{k})\right] \tag{V.1}$$

As mentioned above, the first time-window of $\mathbf{C}(k)$ should be a constant zero since the energy in that time-window is below the required threshold.

1. 8-Point FFT

This process was first tested using an 8-point FFT and one of three digital frequencies ($k = 1,\ k = 2,$ or $k = 3$). The results are captured in Figure 32 through Figure 35. In each figure, the top plot depicts the input signal $IF[n]$ and the bottom plot represents the decompressed IFFT of the design's output. In Figure 32, the digital input

frequency is set so that $k = 1$. The SDR was setup to evaluate that same digital frequency, so the bin's *ROI* vector is $\mathbf{ROI}(1) = [1]$, its vector of FFT indices is $\mathbf{L}(1) = [1]$, and its threshold is set so that $H(1) = 0.0312$.

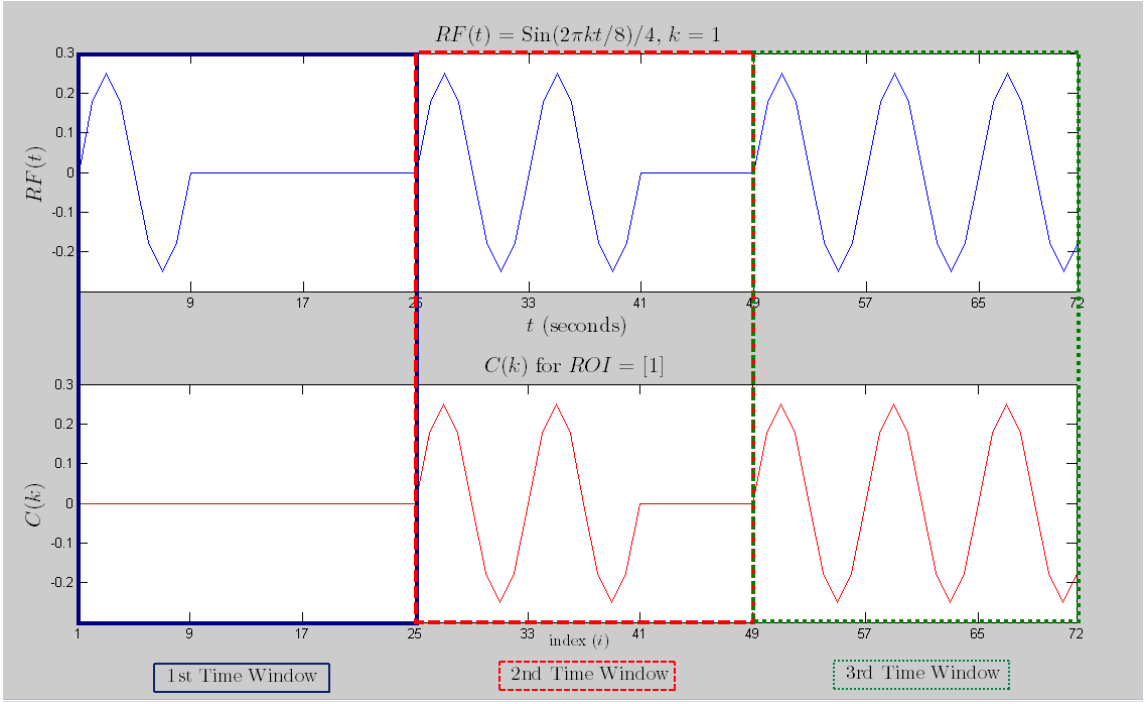


Figure 32. Single Frequency Input Test ($N = 8$ / $k = 1$ / *ROI = 1*).

As explained in the previous section, the established threshold dictates that at least two T_{min} periods of the proper frequency are present, per time-window, for a bin to exceed its energy threshold. If the bin exceeds its threshold then its frequency components should be represented in the IFFT of the decompressed system output, $C(k)$.

The top plot in Figure 32 illustrates that during the first time-window the input signal is only present for one T_{min} period. Since this time-window doesn't contain sufficient energy, the bin's frequency component is not represented in the bottom plot during the first time-window of $C(k)$. For the last two time-windows, since two or more T_{min} periods are present for the input signal, bin requirements are met, and its frequency

components are represented in the associated $C(k)$ time-windows. Based on this analysis, the design compressed a single frequency input signal as intended.

In Figure 33, the digital input frequency is set so that $k = 2$. For this test the SDR was setup to evaluate that same digital frequency, so the bin's *ROI* vector is $\mathbf{ROI}(1) = [2]$, its vector of FFT indices is $\mathbf{L}(1) = [2]$, and its threshold is set so that $H(1) = 0.0312$.

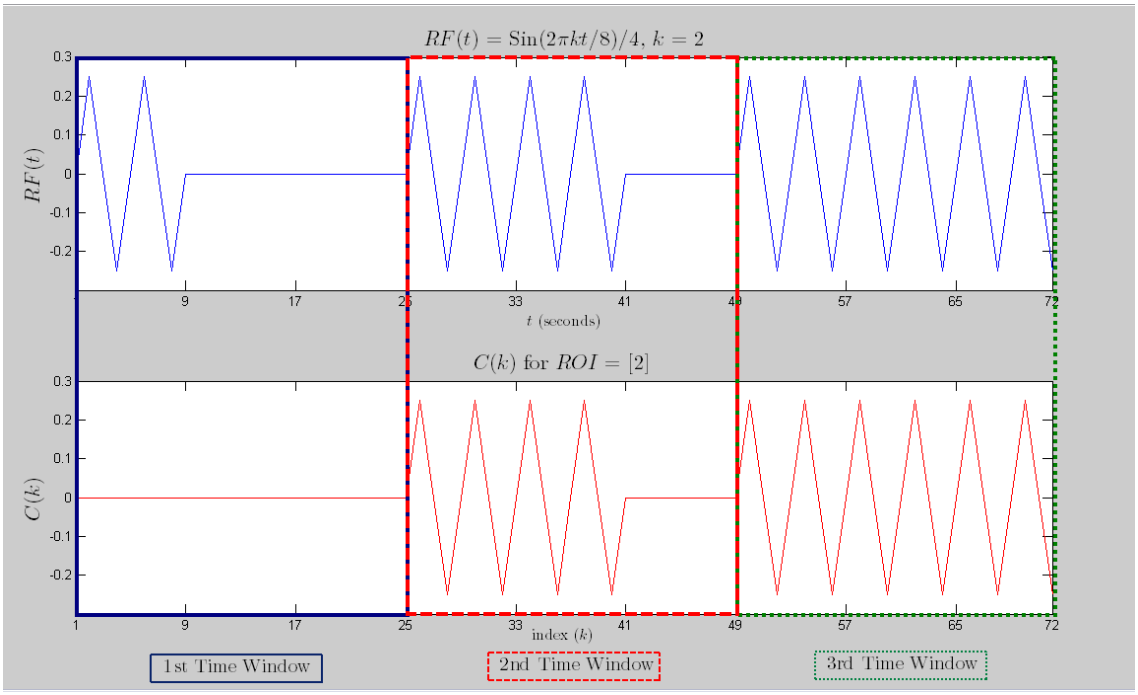


Figure 33. Single Frequency Input Test ($N = 8$ / $k = 2$ / $ROI = 2$).

As in the previous test, the top plot illustrates that during the first time-window the input signal is only present for one T_{min} period. Since this time-window doesn't contain sufficient energy, the bin's frequency component is not represented in the bottom plot during the first time-window of $C(k)$. For the last two time-windows, since two or more T_{min} periods are present for the input signal, the bin requirements are met, and its frequency components are represented in the associated $C(k)$ time-windows. Based on this analysis the design compressed a single frequency input signal as designed.

In Figure 34, the digital input frequency is set so that $k = 3$. For this test the SDR was setup to evaluate that same digital frequency, so the bin's *ROI* vector is $\mathbf{ROI}(1) = [3]$, its vector of FFT indices is $\mathbf{L}(1) = [3]$, and its threshold is set so that $H(1) = 0.0312$.

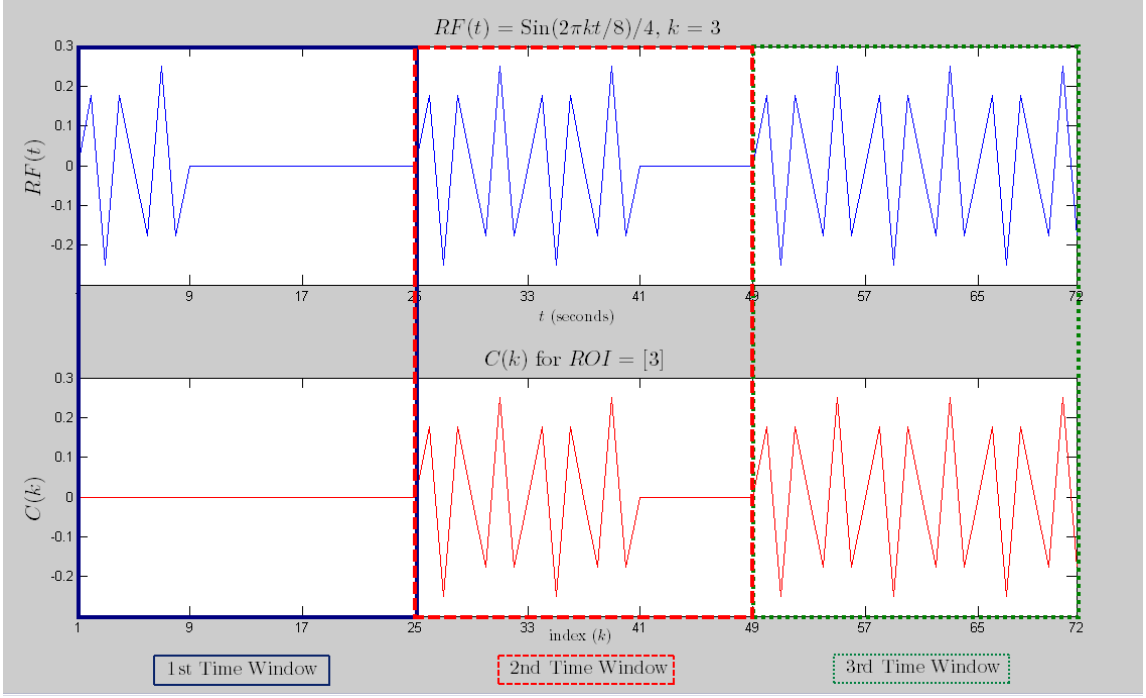


Figure 34. Single Frequency Input Test ($N = 8$ / $k = 3$ / $ROI = 3$).

As in the two previous tests, the top plot illustrates that during the first time-window the input signal is only present for one T_{min} period. Since this time-window doesn't contain sufficient energy, the bin's frequency component is not represented in the bottom plot during the first time-window of $C(k)$. For the last two time-windows, since two or more T_{min} periods are present for the input signal, the bin requirements are met, and its frequency components are represented in the associated $C(k)$ time-windows. Once again, the design compressed a single frequency input signal as designed.

The final single-frequency test was setup slightly different than the first three. Instead of evaluating the digital frequency of the input signal, the system was setup to evaluate another digital frequency. In Figure 35, the digital input frequency is set so that

$k = 1$. For this test the SDR was setup to evaluate $k = 3$, so the bin's *ROI* vector is $\mathbf{ROI}(1) = [3]$, its vector of FFT indices is $\mathbf{L}(1) = [3]$, and its threshold is set so that $H(1) = 0.0312$.

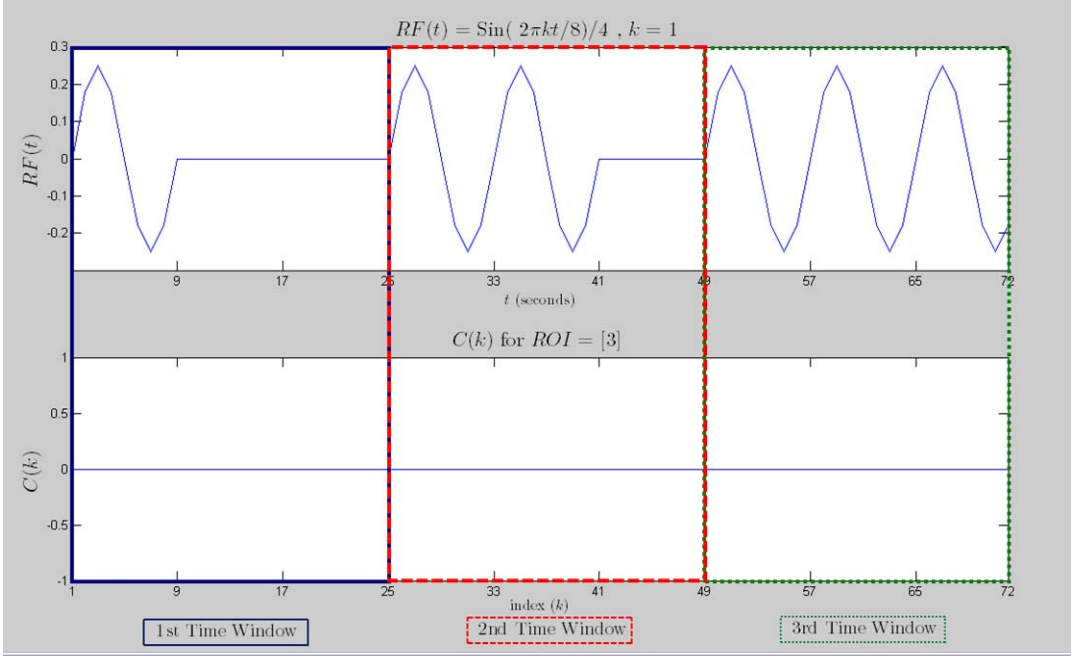


Figure 35. Single Frequency Input Test ($N = 8$ / $k = 1$ / $ROI = 3$).

The input signal is applied in the same fashion as in the first three tests. However, since the design is setup to evaluate a frequency that was not present for any duration of the test period, the defined bin never contains sufficient energy and its frequency component is never represented in $C(k)$. Once again, the design compressed the single frequency input signal as intended.

Based on the four tests described above, the results captured in the Figure 32 through Figure 35 indicate that the SDR responded as intended and is capable of handling single frequency input signals. Although the design operated as expected, it has a performance limitation that should be recognized and considered. All of the development tests were conducted such that the $IF[n]$ signal was a sinusoid with an integer number of cycles per N samples. If the sinusoid did not have an integer number of cycles per N

samples, then the FFT would not have all its energy at one digital frequency, although most of its energy would be in the indices most closely associated with the frequency of the input. Since some of the energy could fall into different bins, results that are very near the threshold might turn out differently if the sinusoid is not an integer number of cycles per FFT period.

For example, if an input signal, $IF[n]$, with a digital frequency of $k = 1.5$ were used in the tests above and the sample frequencies were aligned, signal energy would be distributed over all digital frequencies, but with 51% at $k = 1$ and 36% at $k = 2$. As a result, the result might be different from the first example if the bin energy was close to the bin energy threshold. If this response is unacceptable, the effect can be minimized by increasing the SDR's frequency resolution. As described in Section II.A, this is accomplished by increasing the FFT period. If this is not a viable option or doesn't completely resolve the issue, then the operator must be especially thoughtful as to how bins are defined and thresholds established.

The tests also assumed that the sampled input signal and the SDR shared the same sample frequency. Although the design's sample frequency can be set to ensure this situation exists, it is not a requirement. The $IF[n]$ signal is sampled by an external system and then routed to this design's input ports. If there were a sample rate mismatch, the SDR would need to incorporate interpolation or decimation to address the issue.

2. 1024-Point FFT

The single frequency input test above was also applied to a version of the design that utilized a 1024-point FFT and digital frequencies $k = 1,\ k = 3,$ or $k = 5$. The results of the second round of tests are captured in Figure 36 through Figure 39. Similar to earlier testing, the bin thresholds were established so that in order for a bin to meet its requirement at least two T_{min} periods of the proper frequency had to be applied to the system input during the associated time-window. If the bin meets its threshold requirements then its frequency components should be represented in the IFFT of the decompressed system output, $C(k)$.

For the test results illustrated in Figure 36 the digital input frequency was set so that $k = 1$. The SDR was setup to evaluate that same digital frequency, so the bin's *ROI* vector was $\mathbf{ROI}(1) = [1]$, its vector of FFT indices was $\mathbf{L}(1) = [1]$, and its threshold was set so that $H(1) = 0.0312$.

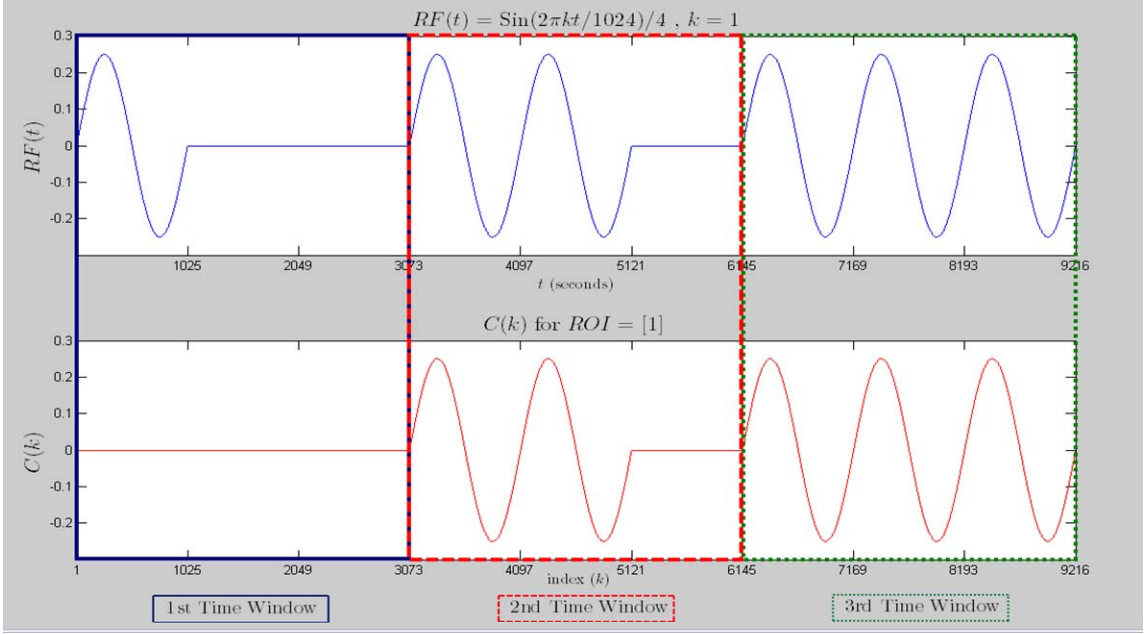


Figure 36. Single Frequency Input Test (N = 1024 / k = 1 / *ROI = 1*).

The top plot in Figure 36 illustrates that during the first time-window the input signal is only present for one T_{min} period. Since this period doesn't contain sufficient energy, the bin's frequency component is not represented in the bottom plot during the first time-window of $C(k)$. For the last two time-windows, since two or more T_{min} periods are present for the input signal, the bin requirements are met, and its frequency components are represented in the associated $C(k)$ time-windows. Based on this analysis the design compressed a single frequency input signal as designed.

In Figure 37, the digital input frequency is set so that $k = 3$. For this test the SDR was setup to evaluate that same digital frequency, so the bin's *ROI* vector was $\mathbf{ROI}(1) = [3]$, its vector of FFT indices was $\mathbf{L}(1) = [3]$, and its threshold was set so that $H(1) = 0.0312$.

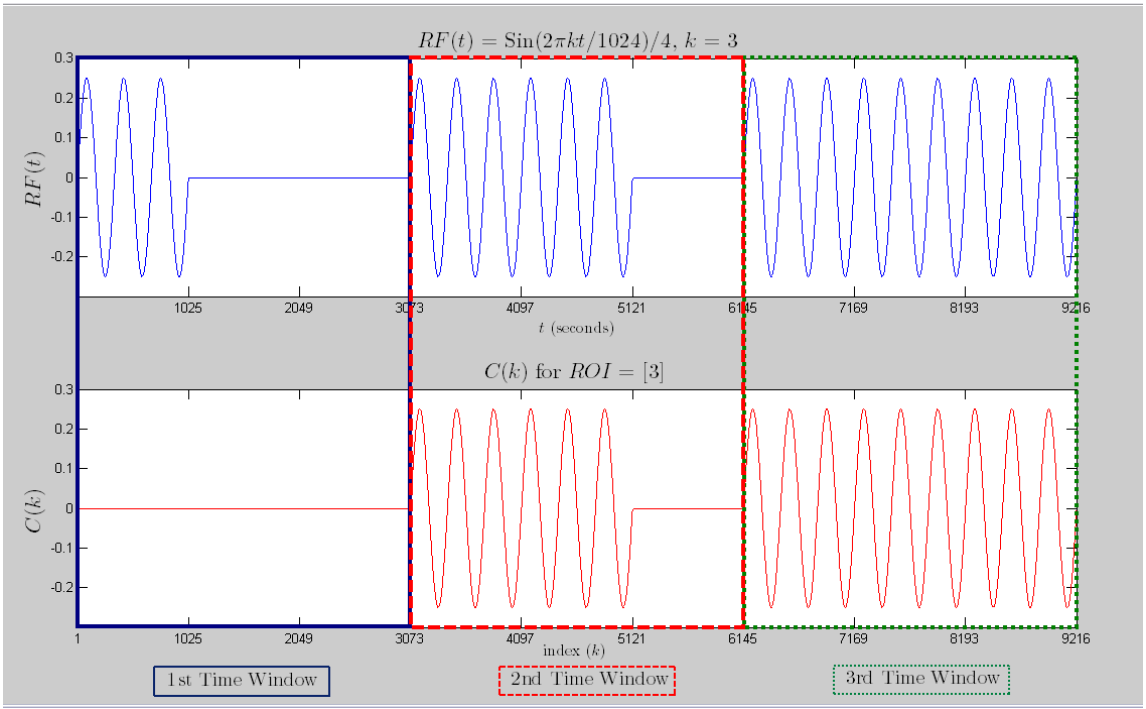


Figure 37. Single Frequency Input Test ($N = 1024$ / $k = 3$ / $ROI = 3$).

As in the previous test, the top plot illustrates that during the first time-window the input signal is only present for one T_{min} period. Since this time-window doesn't contain sufficient energy, the bin's frequency component is not represented in the bottom plot during the first time-window of $C(k)$. However, for the last two time-windows, since two or more T_{min} periods are present for the input signal, the bin requirements are met, and its frequency components are represented in the associated $C(k)$ time-windows. Based on this analysis, the design compressed a single frequency input signal as intended.

In Figure 38, the digital input frequency is set so that $k = 5$. For this test the SDR was set up to evaluate that same digital frequency, so the bin's ROI vector was $\mathbf{ROI}(1) = [5]$, its vector of FFT indices was $\mathbf{L}(1) = [5]$, and its threshold was set so that $H(1) = 0.0312$.

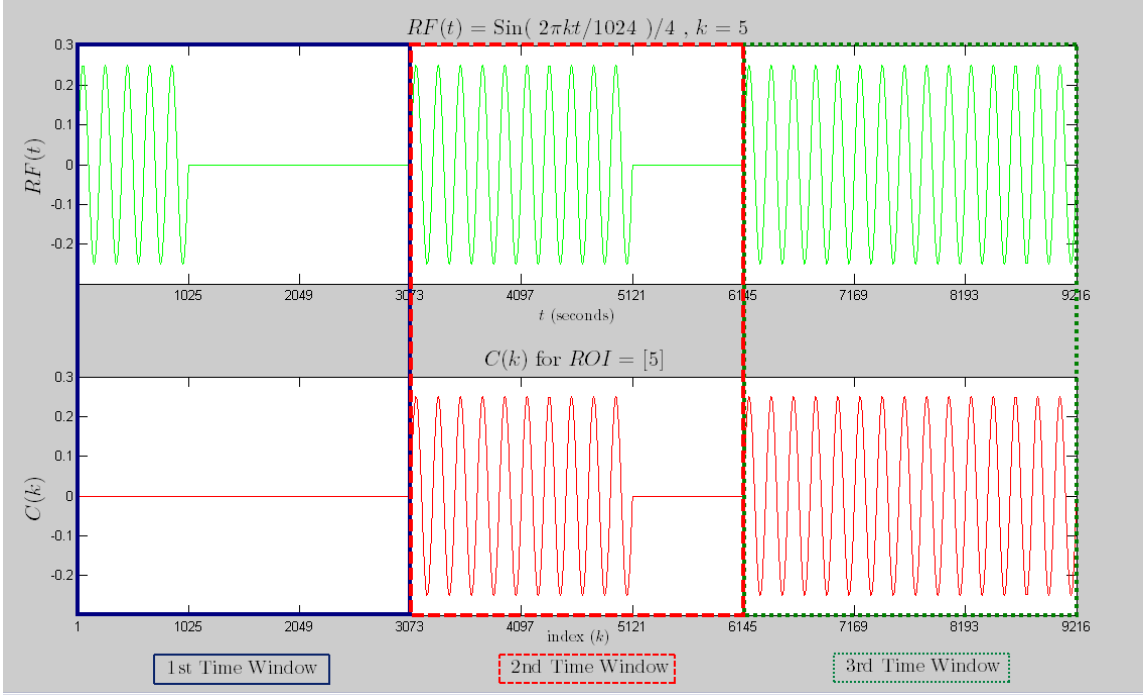


Figure 38. Single Frequency Input Test ($N = 1024$ / $k = 5$ / $ROI = 5$).

As in the two previous tests, the top plot illustrates that during the first time-window the input signal is only present for one T_{min} period. Since this time-window doesn't contain sufficient energy, the bin's frequency component is not represented in the bottom plot during the first time-window of $C(k)$. However, for the last two time-windows, since two or more T_{min} periods are present for the input signal, the bin requirements are met, and its frequency components are represented in the associated $C(k)$ time-windows. Based on this analysis the design compressed a single frequency input signal as intended.

The final single-frequency test was setup so that system did not evaluate the digital frequency of the input signal. In Figure 39, the digital input frequency is set so that $k = 1$. For this test the SDR was setup to evaluate $k = 5$, so the bin's ROI vector was $\mathbf{ROI}(1) = [5]$, its vector of FFT indices was $\mathbf{L}(1) = [5]$, and its threshold was set so that $H(1) = 0.0312$.

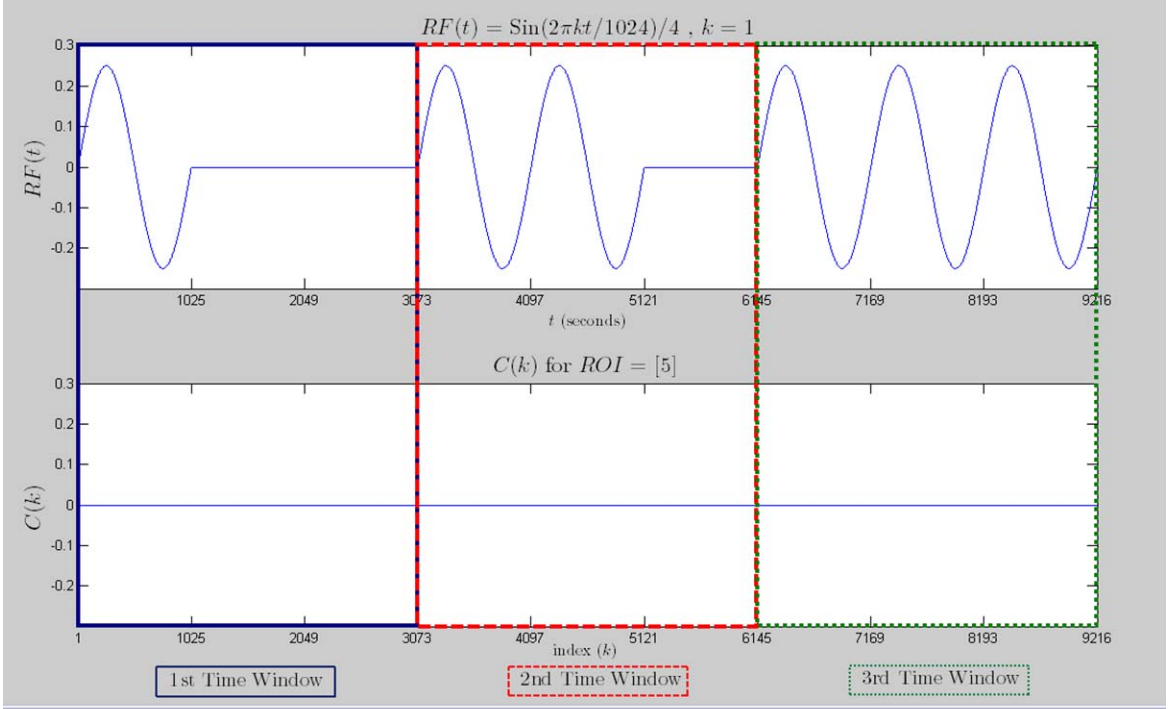


Figure 39. Single Frequency Input Test ($N = 1024$ / $k = 1$ / $ROI = 5$).

The input signal is applied in the same fashion as in the first three tests. However, since the design is setup to evaluate a frequency that was not present for any duration of the test period, the defined bin never contains sufficient energy and its frequency component is never represented in $C(k)$. Once again, the design compressed the single frequency input signal as designed.

Based on the four test described in this section, the results captured in the Figure 36 through Figure 39 indicate that the SDR responded as intended. The frequency alignment issues mentioned in the previous section apply to this test sequence as well.

B. MULTI-FREQUENCY INPUT TEST

In order to test the design's functional response to multi-frequency inputs, a few variations were made to the single frequency input test. Instead of $IF[n]$ containing one frequency component, the signal was reconstituted to include two different single frequency components (IF_1 and IF_2).

$$IF[n] = IF_1[n] + IF_2[n] \quad \text{(V.2)}$$

For a series of tests, the digital frequencies of the two components were set equal to one of three values. Although the duration of IF_1 and IF_2 were identical to each other throughout the test cycle, they were different for each time-window. As in the previous tests, during the first time-window the signals were applied for, T_{min}; during second time-window, they were applied for $2T_{min}$, and during third time-window they were applied for $3T_{min}$.

Another change to the single input test involved adding a second bin to the analysis process. For the new series of tests, the *ROIs* were set to evaluate two of the three optional input frequencies. The bin thresholds $H(1)$ and $H(2)$ were setup in a fashion similar to the single input tests. In order for a bin to meet its threshold, two T_{min} periods of the proper input frequency had to be applied during a time-window. Therefore, during any time-window that either bin's calculated energy meets its energy threshold, the bin's associated frequency component should be represented in the resulting $C(k)$ waveform.

1. 8-Point FFT

The SDR's multi-frequency functionality was tested first using an 8-point FFT and digital frequencies ($k = 1,\ k = 2,$ or $k = 3$). The individual test results are captured in Figure 40 through Figure 43 Although the figures are similar to the results from the single frequency tests, an extra plot was generated for each iteration. The top plots depict the two single frequency components that constitute the $IF[n]$ signal. The middle plots represent the actual input signals $IF[n]$, and the bottom plots represent the decompressed IFFT of the design's output, based on the specified *ROIs*.

For the test results illustrated in Figure 40, the digital input frequencies were $k = 1$ and $k = 2$ and the SDR was setup to evaluate both frequencies in separate bins. The parameters for the first bin were as follows: its range of digital frequencies,

$\mathbf{ROI}(1)=[1]$; the vector of associated FFT indices, $\mathbf{L}(1)=[1]$; and its threshold, $H(1)=0.0312$. For the second bin: $\mathbf{ROI}(2)=[2]$; $\mathbf{L}(2)=[2]$; and $H(2)=0.0312$.

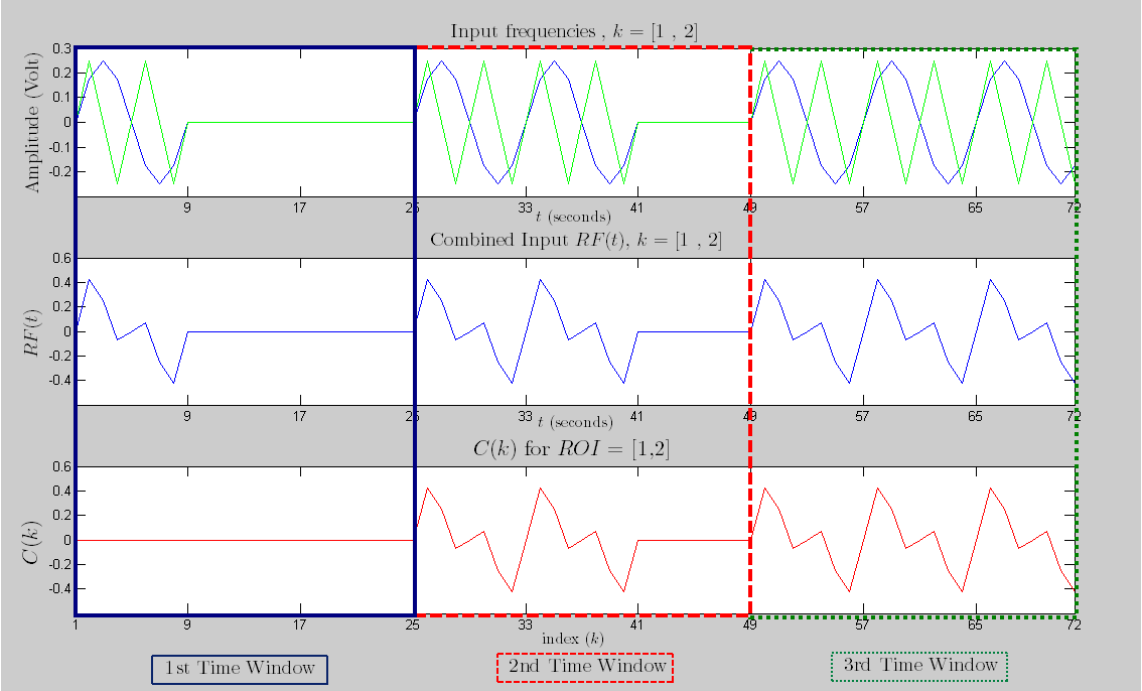


Figure 40. Multi-Frequency Input Test (N = 8 / k = [1,2] / ROI = [1,2]).

The top plot illustrates that during the first time-window both frequency components were present for one T_{min} period. Since this did not satisfy the energy threshold requirements for either bin, neither frequency component is represented in the first time-window of the bottom plot. During the second time-window, the input signal contained two T_{min} periods of each input frequency component. Since this satisfied both bins' energy thresholds, both frequency components are represented in the second time-window of $C(k)$. For the final time-window, the three T_{min} periods of each input frequency exceeded the two bin thresholds. As a result, both frequency components are represented in the last time-window of the output waveform. Based on this analysis the design compressed a multi-frequency input signal as designed.

For the test results illustrated in Figure 41, the digital input frequencies were $k=1$ and $k=3$. The SDR was setup to evaluate the frequencies $k=1$ and $k=2$ in

separate bins. As in the previous test, the two sets of bin parameters were as follows: $\mathbf{ROI}(1)=[1]$; $\mathbf{L}(1)=[1]$; $H(1)=0.0312$; $\mathbf{ROI}(2)=[2]$; $\mathbf{L}(2)=[2]$; and $H(2)=0.0312$.

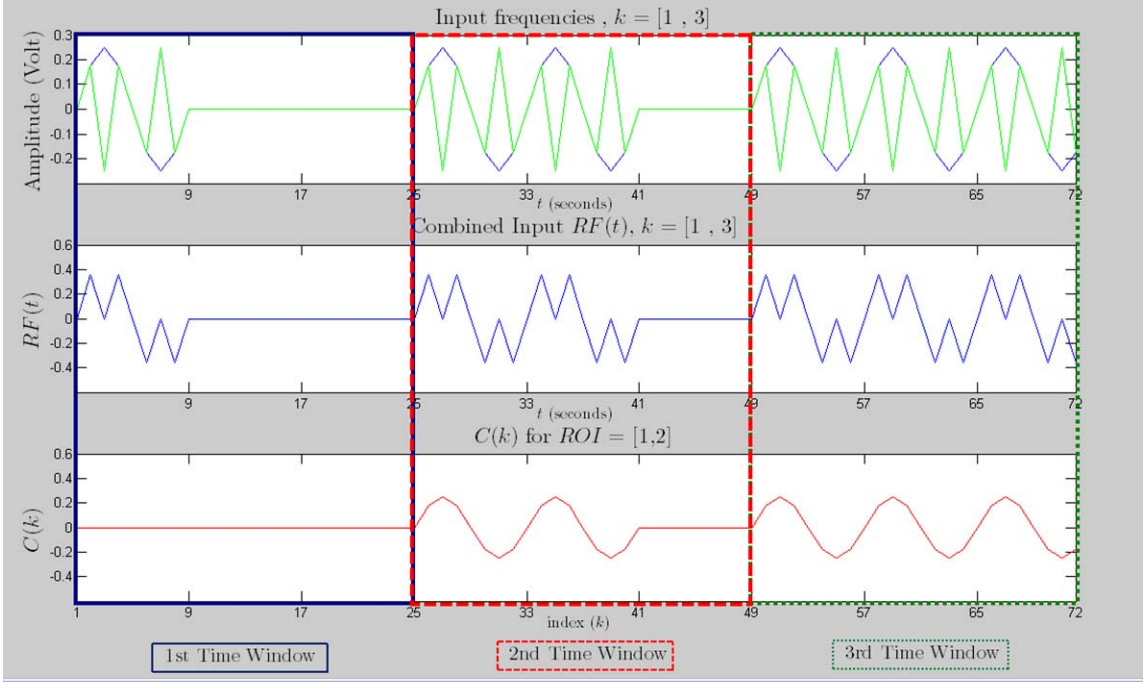


Figure 41. Multi-Frequency Input Test ($N = 8$ / $k = [1,3]$ / $ROI = [1,2]$).

The top plot illustrates that during the first time-window $k=1$ and $k=3$ input frequency components were present for one T_{min} period. During that period, the energy calculated for the first bin did not meet the threshold, so the associated frequency component is not present in the processed output signal. During the second time-window, the input signal consisted of two T_{min} periods of the $k=1$ frequency component. Since this satisfied the first bin's energy threshold, the frequency component is represented in the second time-window of $C(k)$. For the final time-window, there are three T_{min} periods of the $k=1$ input frequency component. This exceeded the first bin's established threshold, so the frequency component is represented in the last time-window of the output waveform. Unlike previous analysis, the second bin was set to evaluate $k=2$, which was never applied to the input. As a result the bin's energy calculations

were constantly zero and the frequency component is never represented in the bottom plot. Based on the analysis of this test, the design compressed a multi-frequency input signal as desired.

For the test results illustrated in Figure 42, the digital input frequencies were $k=1$ and $k=2$. The SDR was setup to evaluate the frequencies $k=1$ and $k=3$ in separate bins. The two sets of bin parameters for the test were as follows: for the first bin $\mathbf{ROI}(1)=[1]$, $\mathbf{L}(1)=[1]$, and $H(1)=0.0312$; and for the second bin $\mathbf{ROI}(2)=[3]$, $\mathbf{L}(2)=[3]$, and $H(2)=0.0312$.

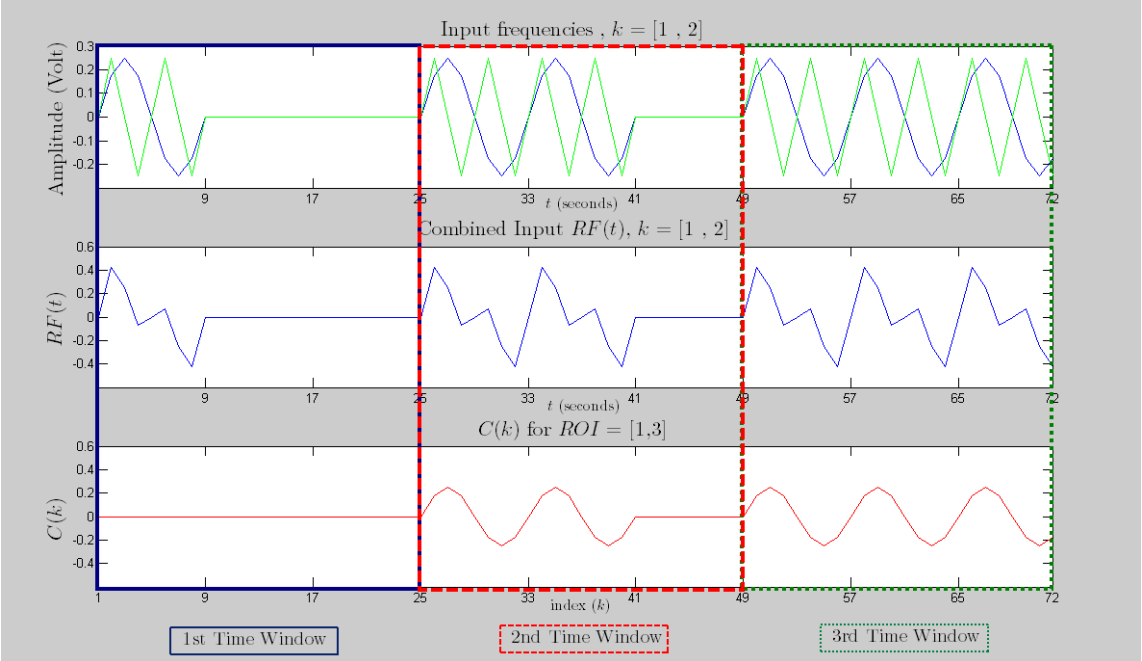


Figure 42. Multi-Frequency Input Test (N = 8 /k = [1,2] / ROI = [1,3]).

The top plot illustrates that during the first time-window $k=1$ and $k=2$ input frequency components were present for one T_{min} period. As in all the previous test the first bin did not meet its threshold during this period, so the associated frequency component is not present in the output during the first time-window. During the last two time-windows, the input signal contained two or more T_{min} periods of the $k=1$ frequency component. Since the associated calculations led to energy values that met

threshold requirements, the frequency component is represented in the last two time-windows of $C(k)$. The frequency component for the second bin was never applied to the input for this test, so its energy calculations were constantly zero. As a result, the frequency component is never represented in the bottom plot. Based on the analysis of this test, the design compressed a multi-frequency input signal as desired.

For the test results illustrated in Figure 43, the digital input frequencies were $k=1$ and $k=3$. However, the SDR was setup to evaluate two unrelated frequencies $k=2$ and $k=4$, in separate bins. The two sets of bin parameters for the test were as follows: for the first bin $\mathbf{ROI}(1)=[2]$, $\mathbf{L}(1)=[2]$, and $H(1)=0.0312$; and for the second bin $\mathbf{ROI}(2)=[4]$; $\mathbf{L}(2)=[4]$; and $H(2)=0.0312$.

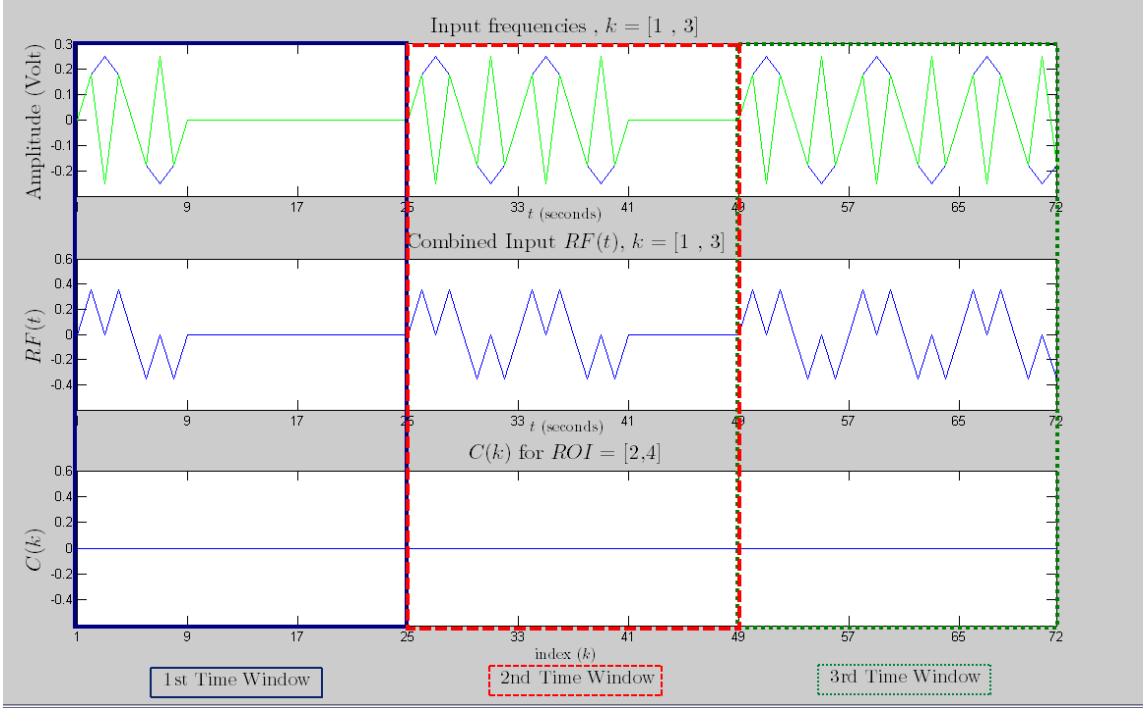


Figure 43. Multi-Frequency Input Test ($N = 8$ / $k = [1,3]$ / $ROI = [2,4]$).

As in previous tests, the top plot illustrates that $k=1$ and $k=2$ input frequency components were applied for varying durations in each of the time-windows. However, since the frequency components defined for the two bins ($k=2$ and $k=4$) were never applied to the input for this test, all energy calculations were equal to zero. As a result,

neither frequency component is represented in the bottom plot. Based on the analysis of this test, the design compressed a multi-frequency input signal as desired.

Based on the four tests described in this section, the results captured in Figure 40 through Figure 43 indicate that the SDR responded as desired and is capable of handling multi-frequency input signals. The frequency alignment issue that was described in the single-frequency testing Section, V.A.1, also impacts this test sequence.

2. 1024-Point FFT

The multi-frequency test was also applied to the 1024-point version of the design. For the second set of tests, digital frequencies $k = 1,\ k = 3,$ or $k = 5$ were used in varying combinations and the results are captured in Figure 44 through Figure 47.

For the test results illustrated in Figure 44 the digital input frequencies were $k = 1$ and $k = 3$ and the SDR was setup to evaluate both frequencies in separate bins. The parameters for the first bin were as follows: its range of digital frequencies, $\mathbf{ROI}(1) = [1]$; the vector of associated FFT indices, $\mathbf{L}(1) = [1]$; and its threshold, $H(1) = 0.0312$. For the second bin: $\mathbf{ROI}(2) = [3]$, $\mathbf{L}(2) = [3]$, and $H(2) = 0.0312$.

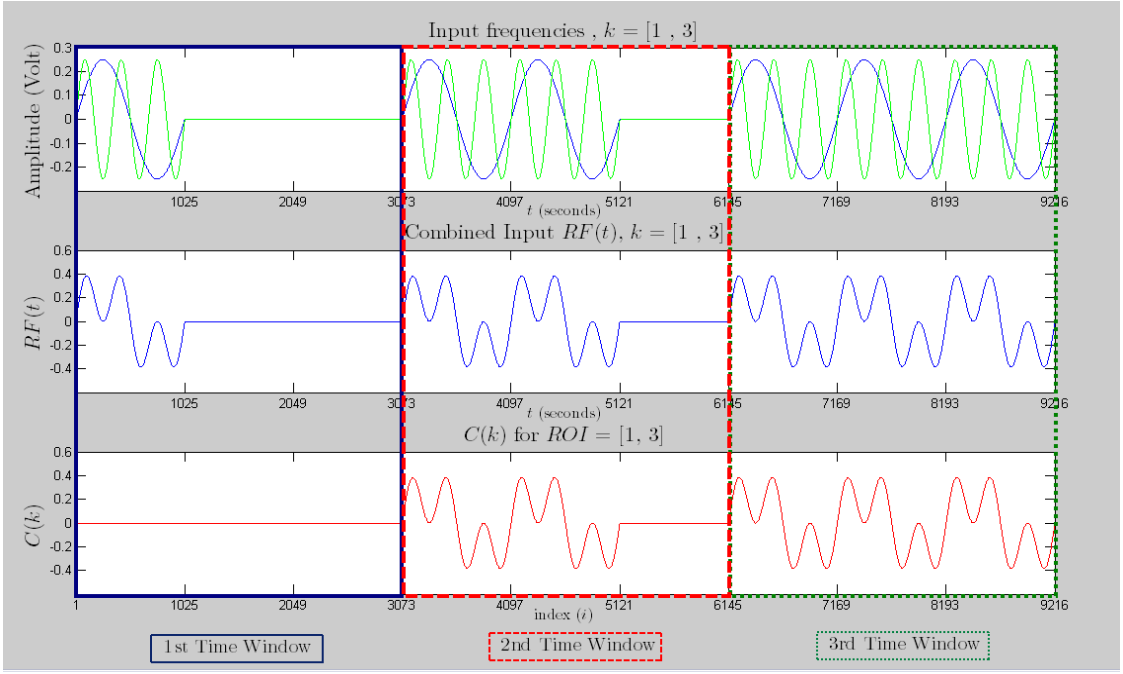


Figure 44. Multi-Frequency Input Test (N = 1024 / k = [1 , 3] / ROI =[1 , 3]).

The top plot illustrates that during the first time-window both frequency components were present for one T_{min} period. Since this did not satisfy the energy threshold requirements for either bin, neither frequency component is represented in the first time-window of the bottom plot. During the second time-window, the input signal contained two T_{min} periods of each input frequency component. Since this satisfied both bins' energy thresholds, both frequency components are represented in the second time-window of $C(k)$. For the final time-window, the three T_{min} periods of each input frequency exceeded the two bin thresholds. As a result, both frequency components are represented in the last time-window of output waveform. Based on this analysis the design compressed a multi-frequency input signal as desired.

For the test results illustrated in Figure 45, the digital input frequencies were once again $k=1$ and $k=3$, but the SDR was setup to evaluate the frequencies $k=1$ and $k=5$ in separate bins. The two sets of bin parameters were as follows: For the first bin $\mathbf{ROI}(1)=[1]$, $\mathbf{L}(1)=[1]$, and $H(1)=0.0312$; For the second bin $\mathbf{ROI}(2)=[5]$, $\mathbf{L}(2)=[5]$, and $H(2)=0.0312$.

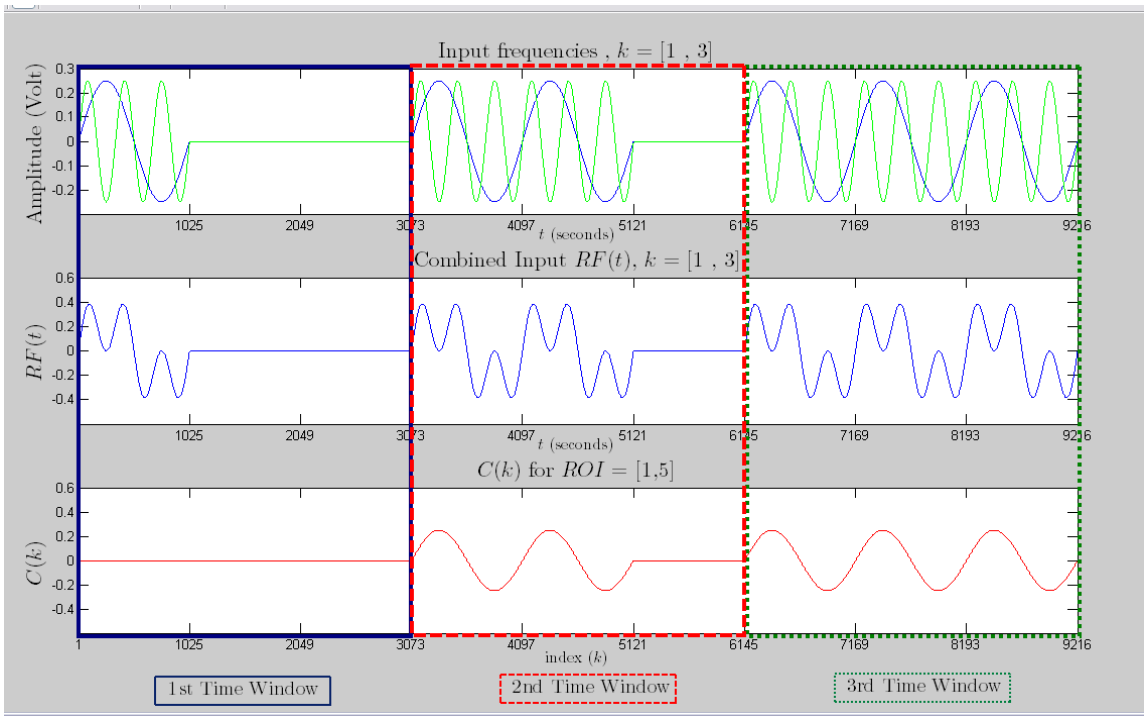


Figure 45. Multi-Frequency Input Test (N = 1024 / k = [1 , 3] / ROI = [1 , 5]).

The top plot illustrates that during the first time-window $k=1$ and $k=3$ input frequency components were present for one T_{min} period. During that period, the energy calculated for the first bin did not meet the threshold, so the associated frequency component is not present in the processed output signal. During the second time-window, the input signal consisted of two T_{min} periods of the $k=1$ frequency component. Since this satisfied the first bin's energy threshold, the frequency component is represented in the second time-window of $C(k)$. For the final time-window, there are three T_{min} periods of the $k=1$ input frequency component. This exceeded the first bin's established threshold, so the frequency component is represented in the last time-window of the output waveform. Unlike previous analysis, the second bin was set to evaluate $k=5$, which was never applied to the input. As a result the bin's energy calculations were constantly zero and the frequency component is never represented in the bottom plot. Based on the analysis of this test, the design compressed a multi-frequency input signal as desired.

For the test results illustrated in Figure 46 the digital input frequencies were $k=3$ and $k=5$ and the SDR was setup to evaluate both frequencies in separate bins. The two sets of bin parameters were as follows: For the first bin $\mathbf{ROI}(1)=[3]$, $\mathbf{L}(1)=[1]$, and $H(1)=0.0312$; For the second bin $\mathbf{ROI}(2)=[5]$, $\mathbf{L}(2)=[5]$, and $H(2)=0.0312$.

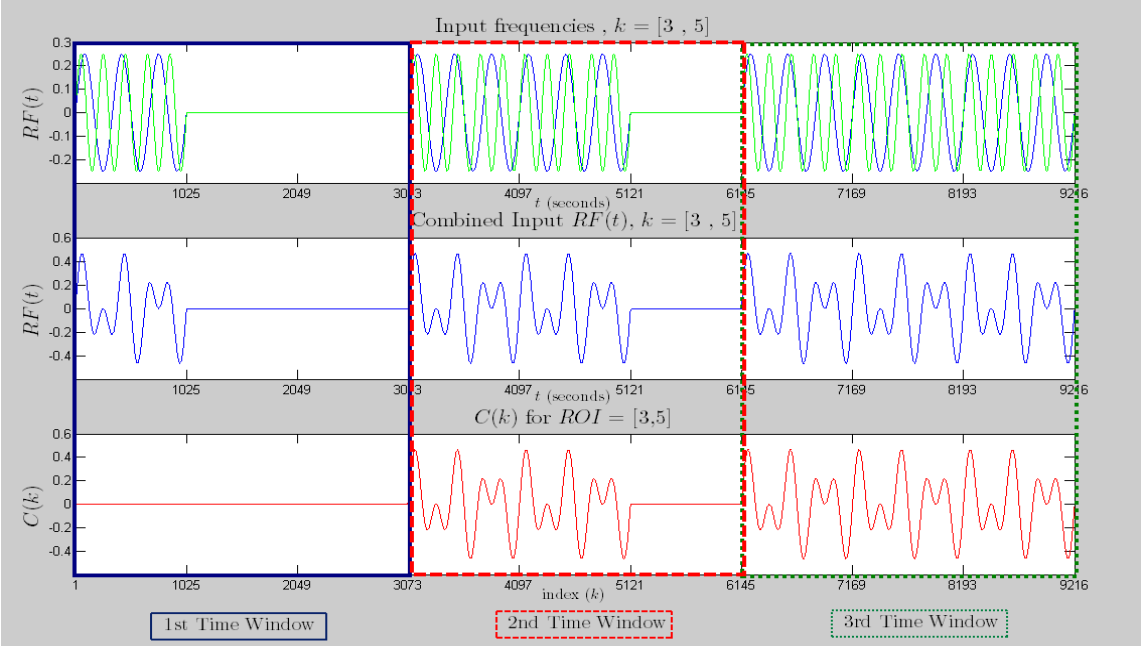


Figure 46. Multi-Frequency Input Test (N = 1024 / k = [3 , 5] / ROI =[3 , 5]).

The top plot illustrates that during the first time-window both frequency components were present for one T_{min} period. Since this did not satisfy the energy threshold requirements for either bin, neither frequency component is represented in the first time-window of the bottom plot. During the second time-window, the input signal contained two T_{min} periods of each input frequency component. Since this satisfied both bins' energy thresholds, both frequency components are represented in the second time-window of $C(k)$. For the final time-window, the three T_{min} periods of each input frequency exceeded the two bin thresholds. As a result, both frequency components are represented in the last time-window of the output waveform. Based on this analysis the design compressed a multi-frequency input signal as desired.

For the test results illustrated in Figure 47, the digital input frequencies were again $k = 3$ and $k = 5$, but the SDR was setup to evaluate $k = 2$ and $k = 4$, in separate bins. The two sets of bin parameters were as follows: For the first bin $\mathbf{ROI}(1) = [2]$, $\mathbf{L}(1) = [2]$, and $H(1) = 0.0312$; For the second bin $\mathbf{ROI}(2) = [4]$, $\mathbf{L}(2) = [4]$, and $H(2) = 0.0312$.

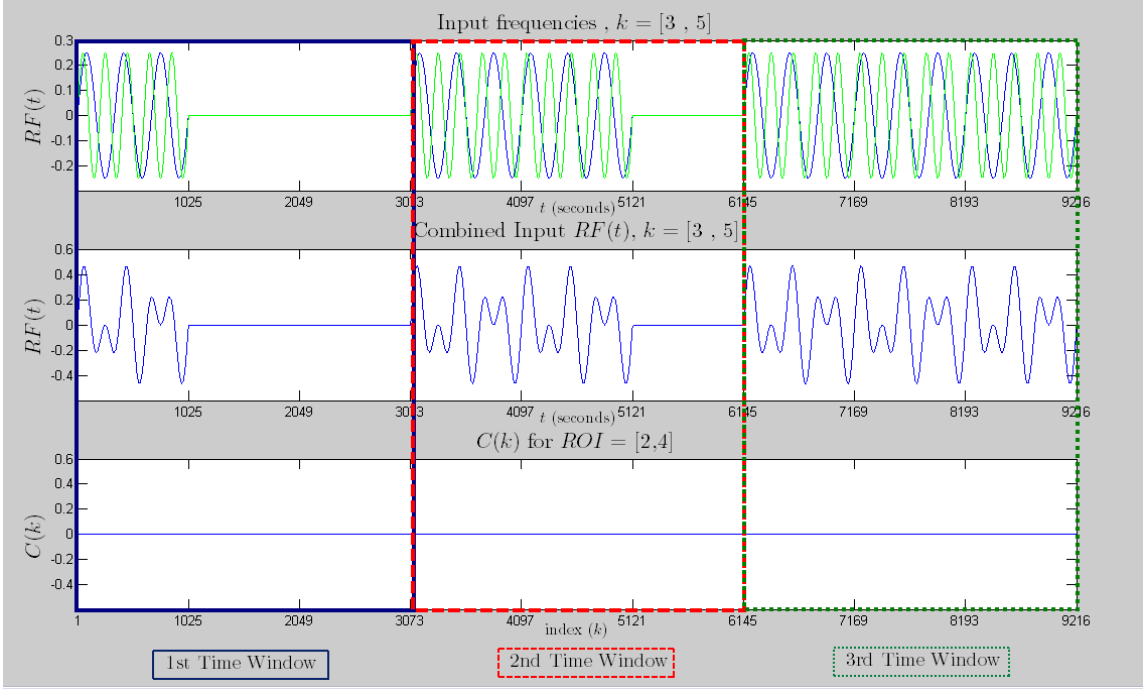


Figure 47. Multi-Frequency Input Test (N = 1024 / k = [3 , 5] / ROI = [1 , 4]).

As in previous tests, the top plot illustrates that $k = 3$ and $k = 5$ input frequency components were applied for varying durations in each of the time-windows. However, since the frequency components defined for the two bins ($k = 2$ and $k = 4$) were never applied to the input for this test, all energy calculations were equal to zero. As a result, neither frequency component is represented in the bottom plot. Based on the analysis of this test, the design compressed a multi-frequency input signal as desired.

Based on the four test described in this section, the results captured in Figure 44 through Figure 47 indicate that the SDR responded as desired and is capable of handling multi-frequency input signals.

C. MEMORY COMPENSATION TEST

The last functional test requires restricting the available output memory and verifying that the system is able to adjust the data output scheme accordingly. In order to simulate restrictions on output memory, the multi-frequency test was used with a few changes to design parameters: Capacity for the final output memory was reduced to hold

only 32 data points; the read-enable control signal, rE_final, was set to zero for prolonged periods so that no data could be read from memory; the memory threshold, U, was set to 25%; the $ROIs$ used for analysis were matched to the input frequency components; and finally, input sinusoids were present throughout the test period so that both bins would exceed their established energy thresholds during all three time-windows.

The memory compensation test was run three times using the 1024-point version of the design. Results from each memory capacity test include a total of six plots, and are illustrated in Figure 48 through Figure 50. The top plots depict the two single frequency components that constitute the $IF[n]$ signal. The second plots represent the actual input signal $IF[n]$. The third plot represents the status of the read-enable signal, during the associated time-window input period. Since there is a delay ($proc\_del$) between the input signal and storage of related data points, the indices (τ) for these plots are adjusted to visually align with the relevant input and data storage periods.

$$\tau = t + proc\_del \tag{V.3}$$

For the 1024-point version of the design, the $proc\_del$ =3072. The next two plots are also functions of τ. The fourth plot represents the output memory capacity, $\%full$, relative to a time-window input period. The fifth plot depicts the status of the $pri\_fl(w)$ control signal, which determines the design's operating mode; described in Section IV.B.3.f. The sixth and last plot reveals the decompressed IFFT of the system's output, $C(k)$. Under normal operating conditions, the system should store the data points for both bins. However, when memory capacity is limited with respect to U, the system should only store data for the first bin.

The first iteration of the memory compensation test was setup as a baseline to demonstrate the system's output in the default operating mode. For this test, the digital input frequencies were $k = 1$ and $k = 3$ and the SDR was setup to evaluate both frequencies in separate bins. The parameters for the first bin were as follows: the range of

digital frequencies, $\mathbf{ROI}(1) = [1]$; the vector of associated FFT indices, $\mathbf{L}(1) = [1]$; and its threshold, $H(1) = 0.0156$. For the second bin: $\mathbf{ROI}(2) = [3]$, $\mathbf{L}(2) = [3]$, and $H(2) = 0.0156$. Another set of parameters that were important to this test were the default, s, and alternate, s_{pri}, number of bins that should processed per bin set. For the design's normal operating mode the default was set so that $s = 2$, and for situations when memory capacity exceeded the established threshold the alternate was set so that $s_{pri} = 1$. The system and bin parameters defined for this test were used for all three memory compensation tests.

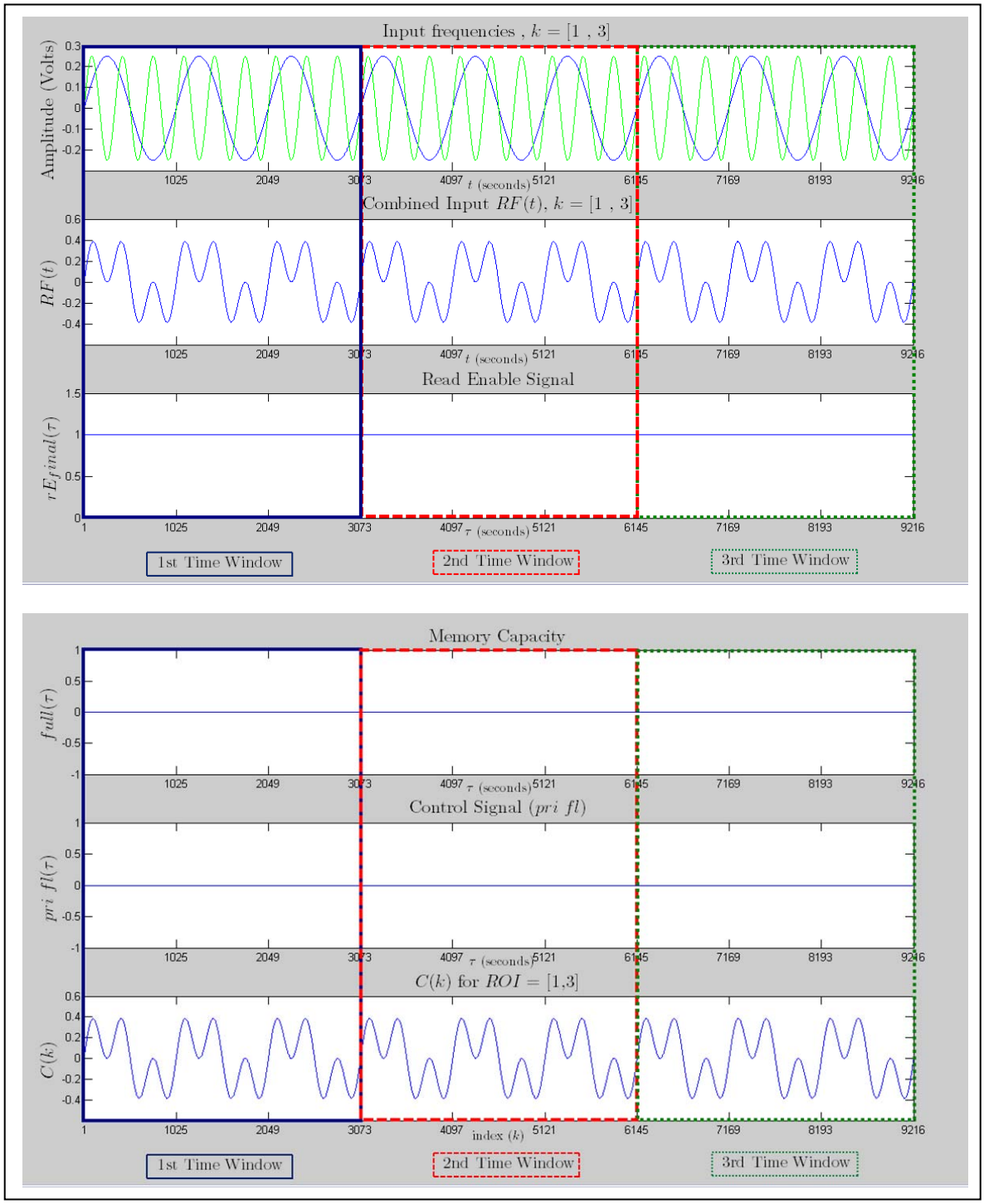


Figure 48. SDR Output Memory Compensation (Test 1).

The top two plots illustrate the input signal $IF[n]$ and its frequency components. Since there are three T_{min} periods of both frequencies in each time-window, both bins meet their energy thresholds for the entire test period. This means that if the $pri\_fl(w)$ control signal is zero during a time-window, then both input frequency components should be represented in the associated time-window of the final output waveform, $C(k)$.

Based on the plots in Figure 48, the $pri\_fl(w)$ was zero for the entire test period. As expected, both frequency components were represented in the appropriate plots and the system performed as desired. Plots three through five are the key to understanding the design's mechanics. Since the output memories read enable control signal, rE_final, was one for entire test period the data stored never filled more than one of the thirty-two available memory locations. This situation ensured that memory capacity, $\%full$, never exceeded the storage threshold, U. As a result, the $pri\_fl(w)$ was zero for the entire test period and the input signal, $IF[n]$, and final output waveform, $C(k)$, were a match.

For the second test, the system and bin parameters were identical to that of the first test: The digital input frequencies components were $k=1$ and $k=3$; the first bin's parameters were $\mathbf{ROI}(1)=[1], \mathbf{L}(1)=[1], H(1)=0.0156$; the second bin's parameters were $\mathbf{ROI}(2)=[3]$, $\mathbf{L}(2)=[3]$, $H(2)=0.0156$; and the number of bins analyzed per bin set were a function of $s=2$ and $s_{pri}=1$. The differences in this test are a function of changes to the output memory's read enable control signal, rE_final. The test results are captured in Figure 49.

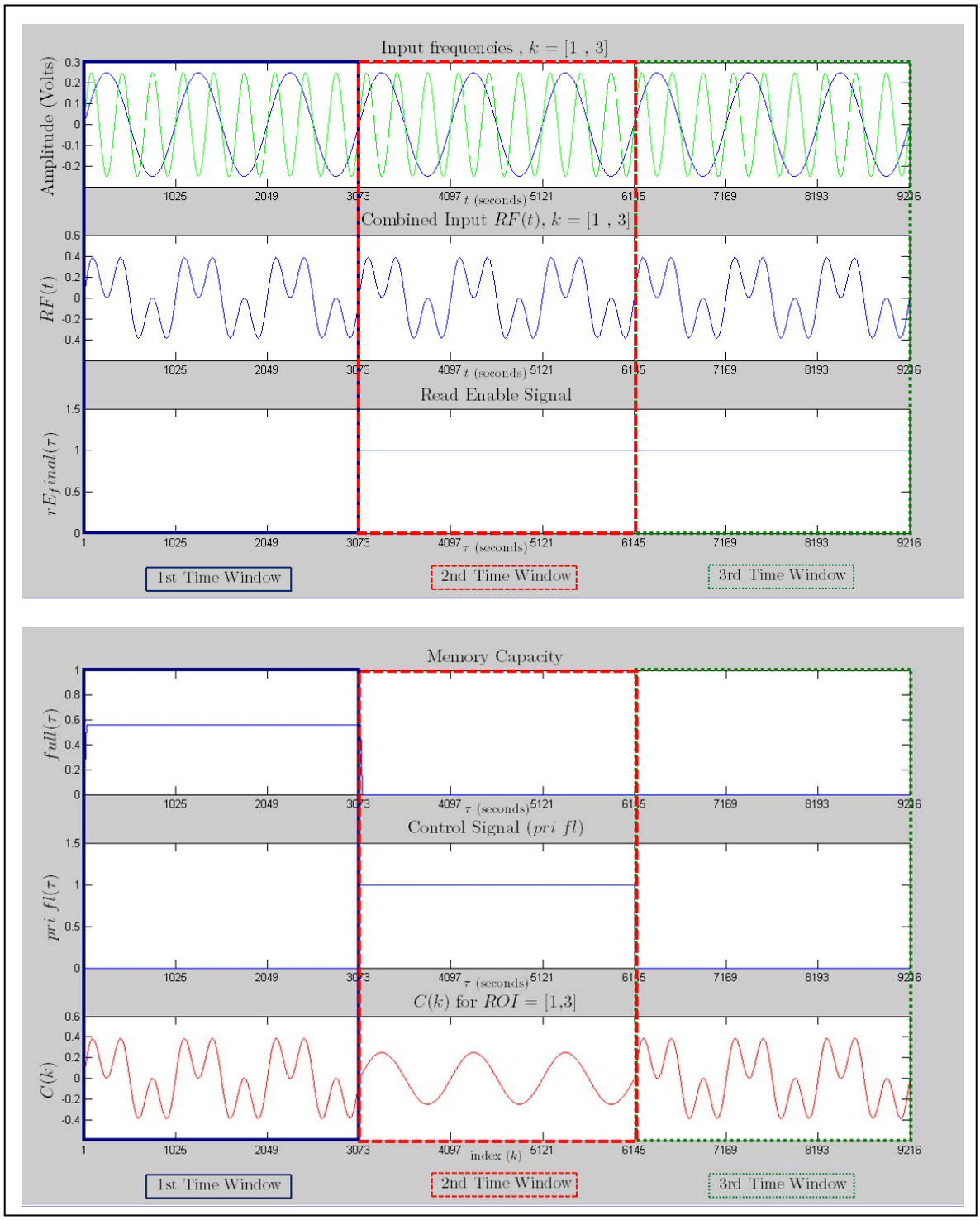


Figure 49. SDR Output Memory Compensation (Test 2).

As discussed in the previous test, both bins meet their energy thresholds for all three time-windows. This means that for any time-window that the design is operating in its default mode, $pri\_fl(w) = 0$, the input signal shown in the second plot should be identical to the processed output signal shown in the bottom plot. However, for any time-window in which $pri\_fl(w) = 1$, only the $k = 1$ frequency component should be represented in the output waveform's associated time-window.

As illustrated in Figure 49, the rE_final signal was zero for the first time-window and was then changed to one for the last two time-windows. As a result, the compressed FFT data and bin set header for the first time-window occupied more than 25% of the available output memory. This situation changed the system's operating mode for the second time-window, which is indicated by the fact that $pri\_fl(2) = 1$ for that period. Accordingly, the second time-window of $C(k)$ only represents the $k = 1$ frequency component. Since the rE_final signal remained high for the duration of the test period, the memory usage, $\%full$, remained below the threshold for the last time-window. As a result, the $pri\_fl(3)$ was zero for the period, the system returned to its default operating mode, and both frequency components were represented in the final output waveform.

For the third test, the system and bin parameters were once again identical to that of the first test: The digital input frequencies components were $k = 1$ and $k = 3$; the first bin's parameters were $\mathbf{ROI}(1) = [1], \mathbf{L}(1) = [1], H(1) = 0.0156$; the second bin's parameters were $\mathbf{ROI}(2) = [3]$, $\mathbf{L}(2) = [3]$, $H(2) = 0.0156$; and the number of bins analyzed per bin set were a function of $s = 2$ and $s_{pri} = 1$. The differences in this test are again a function of changes to the output memory's read enable control signal, rE_final. The test results are captured in Figure 50 .

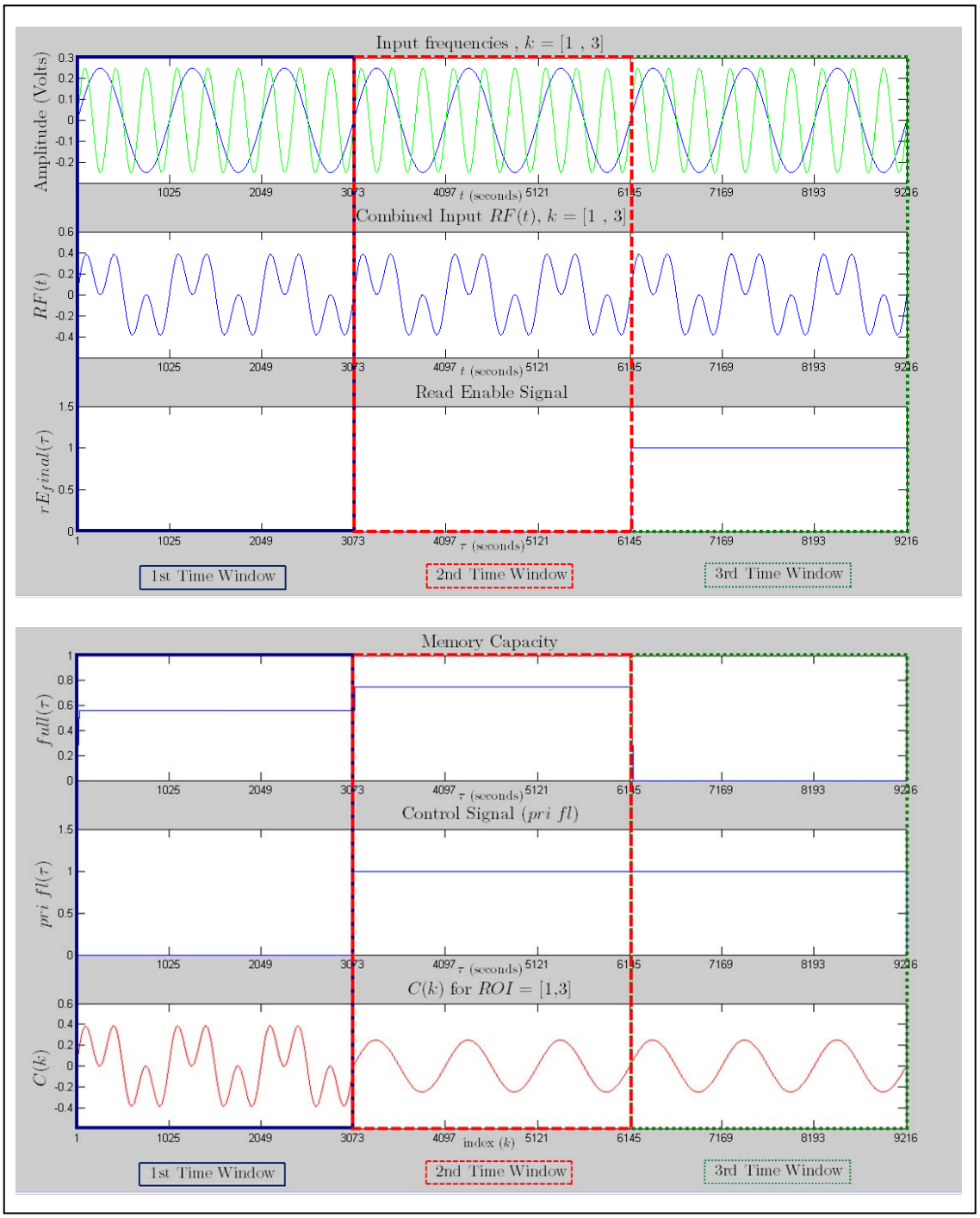


Figure 50. SDR Output Memory Compensation (Test 3).

As in the two previous tests, both bins met their energy thresholds for all three time-windows. Therefore, for any time-window in which the design is operating in its default mode, $pri\_fl(w) = 0$, the input signal shown in the second plot should be identical to the processed output signal shown in the bottom plot. However, for any time-window in which $pri\_fl(w) = 1$, only the $k = 1$ frequency component should be represented in the associated period of the final output waveform.

As illustrated in Figure 50, the rE_final signal was zero for the first two time-windows and was then changed to one for the third. As a result, the compressed FFT data and bin set header for the first two time-windows occupied more than 25% of the available output memory. This situation changed the system's operating mode for the last two time-windows, which is indicated by the fact that $pri\_fl(2) = pri\_fl(3) = 1$. Accordingly, in the last two time-windows of the output waveform, $C(k)$, only the $k = 1$ frequency component is represented.

Based on the three tests described in this section, the results captured in Figure 48 through Figure 50 indicate that the SDR responded as desired and is able to adjust its operational mode according to the available output memory.

D. LESSONS LEARNED

During the testing process, two notable points were discovered. First, parameter changes must be made to key elements when scaling the design's FFT period. Second, a digital system's machine epsilon must be considered when processing its output data. The concept will be described in Section V.D.2.

1. Scaling Considerations

The SysGen FFT module is the most obvious place to start when considering elements affected by scaling the points per FFT. In order to adjust the number of data

points associated with an FFT period, N, the module provides a dropdown menu (Number of sample points) with all the available options. In Figure 51, the parameter is boxed in red.

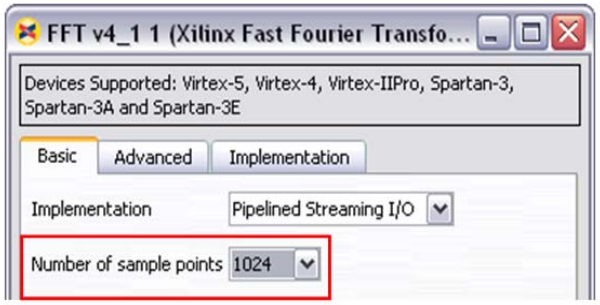


Figure 51. FFT Period (Parameter).

As the FFT period is changed, the $1/N$ scaling factor in the FFT definition, Eq. (II.4), must be accounted for in the SDR design. As mentioned in Section IV.A.1, the SysGen FFT module's output was not scaled so the SysGen Scale modules, boxed in Figure 52, were used instead. The modules scale numerical values by powers of two, so to represent a value of $1/8 = 2^{-3}$, an operator would use the module's interface to set a value of -3. For this design, the value entered in the Scale modules is referred to as N_{scale}. Point being, in order to properly implement the $1/N$ scaling, N_{scale}, must be manually adjusted in both Scaling modules where

$$N_{scale} = -\log_2(N). \tag{V.4}$$

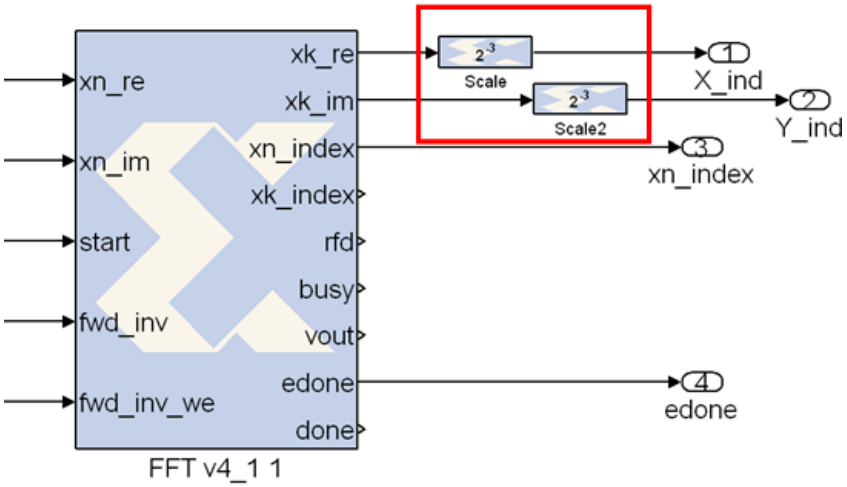


Figure 52. N_{scale} (Parameter).

Memory capacity is another element of the design that can be greatly affected by a change in the FFT period. As N increases, so do the memory requirements. As a result, considerations must be made regarding the appropriate memory depth for each of the design's storage components. The design uses 11 storage devices to work with the system's analysis data. A complete list is provided in Table 7.

Table 7. Storage Devices (Design Analysis Data).

Storage Device Type	Control Module
FIFO (Time Wind)	Energy (Time Window)
Dual Port RAM (Freq Wind)	Energy (Frequency Window)
Dual Port RAM (Real Data)	Temporary FFT Data Storage
Dual Port RAM (Imag Data)	Temporary FFT Data Storage
FIFO (*ROI* - Header)	Header Generation
FIFO (w - Header)	Header Generation
FIFO ($anal\_qty$ - Header)	Header Generation
FIFO (*ROI* - Temp Data)	Temp. Data Read Control
FIFO (w - Temp Data)	Temp. Data Read Control
FIFO ($anal\_qty$ - Temp Data)	Temp. Data Read Control
FIFO (Final Output)	Output Format

The design's addressing schemes can also be affected by changes to the FFT period. The system utilizes the four addressing algorithms listed in Table 8 to manage data storage requirements. Each of them generates an address based on two binary words. The least significant word of each algorithm can range from zero to $N-1$. This fact makes it critically important that each algorithm's least significant word is defined with at least $\log_2(N)$ bits.

Table 8. Address Generation Algorithms.

Addressing Algorithm	Control Module
wE_time_win	Write Enable (Time Window)
rE_freq_win	Read Enable (Freq Window)
wE_temp_fft	Temporary FFT Data Storage
rE_tmp	Temporary Data Read Control

2. Machine Epsilon Considerations

There are varying definitions of machine epsilon, but for fixed-point numbers, they all involve determining the smallest positive number that a digital system can recognize and generate. When working with fixed-point designs, machine epsilon is a function of the number of binary digits that follow the decimal point. For example, if a design uses a Fix_4_3 arithmetic word (fixed point, four total bits, three after the decimal) then the smallest value that can be represented is $0.001_2 = \left(2^{-3}\right)_{10} = 0.1250_{10}$. However, if the same design were to use a Fix_6_5 arithmetic word (fixed point, six total bits, five after the decimal) then the smallest value that could be represented is $0.00001_2 = \left(2^{-5}\right)_{10} = 0.03125_{10}$. When combined with rounding or truncating, machine epsilon can lead to numerical calculations that should mathematically equal zero but result in other values. Such is the case with all digital designs.

To address the issue in the final design, two different solutions were investigated. First, the algorithm used to process the design's output was altered so that it would discard any numerical value (real or imaginary) that fell below the design's machine epsilon. In this design, the machine epsilon is determined by the fixed-point binary format, Fix_D_D-1, that is used for the inputs to the SysGen FFT module, as described in Figure 53. Second, the design was altered and tested with greater binary

resolution. This was done by increasing the number of bits used in each of the system's calculations. All the components affected by changing the binary format are listed in Table 9 but as mentioned above the major catalyst for change are the data formats used in the IF_{real} input gateway and the *Imag* signals illustrated in Figure 53.

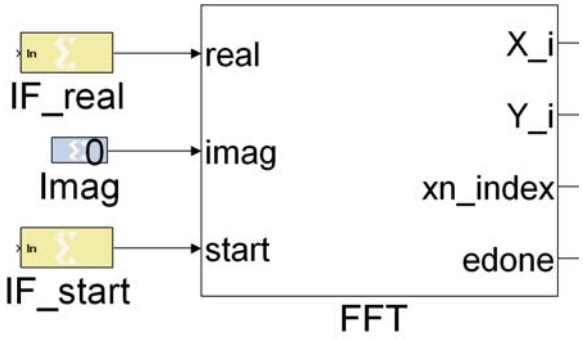


Figure 53. FFT Inputs Requiring Change.

The two gateways feed directly into the SysGen FFT module and their binary resolutions determine the FFT module's output resolution. For instance, during the first round of testing the gateways' binary format was Fix_12_11 and the FFT module's output format was Fix_16_11. For the second round of testing, the gateways' binary format was changed to Fix_24_23 and the resulting FFT output format was Fix_28_23. This general change in output format ultimately affects each of the design elements in Table 9. Therefore, the binary formatting parameters in each must be appropriately adjusted.

Table 9. Modules Affected by Changes to Numerical Binary Format.

Design Element	Control Module	Considerations
Gateway In (IF_{real})	NA	Formats the real component of the input to the FFT module and determines the modules output format. (Based on system interface)
Imag	NA	Formats the imaginary component of the input to the FFT module and determines the modules output format. (Based on system interface)
AddSub	Energy (Time Window)	Input to the module comes from the Energy (FFT Index) module. Therefore, the AddSub module should have an equal number of bits after the decimal point. To determine the number of bits before the decimal, an operator should estimate the maximum value that could be calculated in a $\mathbf{E}_{\mathbf{t}ine}\left(w\right)$ vector.
d_in_b (Temp Data)	Temp. FFT Data Storage	Must match the output format from the FFT module.
d_in_b (Freq Wind)	Energy (Freq. Window)	Must match the output format from the FFT module.
Accumulator_1	Accumulator Control (Freq. Window)	'Output Precision: Number of Bits' parameter must match the 'Binary Point' parameter of the SysGen AddSub module used in the Energy (Time Window) control module
Accumulator_2	Accumulator Control (Freq. Window)	'Output Precision: Number of Bits' parameter must match the 'Binary Point' parameter of the SysGen AddSub module used in the Energy (Time Window) control module

Increasing the design's binary resolution results in reduced machine epsilon, but it does not guarantee that calculations will result in the expected zero values. Therefore, adjusting the system's fixed-point formats is only useful if the design's application requires more fidelity.

E. SUMMARY

This chapter explains the tests used to validate the SDR's functional operations and scaling requirements. Three functional elements of the system's response were verified: single frequency input; multi-frequency input; and memory compensation. Based on the tests results, the SDR performed as desired. In order to test the design's scaling requirements, it was built and tested with two different FFT periods, $N = 8$ and $N = 1024$. Components affected by the changes in FFT period are captured in Table 7, Table 8, and Table 9. Another benefit of the testing process was that the effects and workarounds for machine epsilon were realized. Chapter VI summarizes the body of work captured in the thesis and then provides recommendations for follow-on work.

VI. CONCLUSION

This chapter provides a summary of the thesis and reviews the major concepts that influenced the development of the FPGA-based SDR design. The chapter also contains recommendations regarding further research and updates to the final design.

A. CONCLUSION

This thesis was conceived to help mitigate the restrictions imposed on FPGA-based communications radio designs by external IO bandwidth mismatches. The goal was to design an FPGA-based SDR that could compress sampled wideband IF signals based on reprogrammable parameters. The design was developed around the concept of independent, operator-defined time-frequency bins and evaluation of the energy in each bin. Although the design concept incorporated bins with varying time-window periods, the final design was simplified so that each bin used the same period.

Xilinx's System Generator software was utilized to develop and test the behavioral definition of the design. The tool was also used to synthesize the design, perform the place-and-route functions, and generate the .bin file that provides the FPGA's configuration information. The development tool provided a layer of abstraction that reduced the requirement for in-depth of knowledge with respect to HDL coding. If the design required management of internal hardware clocks, then the ability to code in a HDL would have been more critical.

The SDR was developed for a Virtex-4 FPGA architecture. While this can potentially affect its portability, the SysGen FFT v4.1 module was the only component used that is not backward compatible to the Virtex-1. This issue can be resolved by using a different version of the SysGen FFT module and then making minor changes to parameters in the pwr_time control algorithm.

Two versions of the algorithm were built and tested. The first utilized an 8-point FFT, which simplified analysis efforts. The second version utilized a 1024-point FFT and helped verify the requirements for scaling the design. Both versions were tested using

single and multi-frequency input signals, without restrictions on the output memory. This ensured the basic compression scheme operated properly. Then, the 1024-point algorithm was used to verify the design automatically adjusted its operations based on the available storage capacity. Testing provided valuable insights regarding the effects and workarounds for machine epsilon. It also verified the design's desired functional operations.

Although the final design operated as expected, it has performance limitations that should be recognized and considered. All of the development tests were conducted such that the digital input frequencies and the defined bin frequencies were an exact match. If frequencies that did not match the FFT window, i.e., frequencies that did not have an integer number of cycles per FFT window, were used there would be a slight smearing effect in the frequency domain. It is assumed that this effect is minor, and would not significantly impact the efficacy of the algorithm. The tests also assumed the input signal and the SDR shared the same sample frequency. If this were not the case, the implementation could correct for this with interpolation or decimation, or similar multirate signal processing.

B. RECOMMENDATIONS

The research and development process for this thesis resulted in a simple, FPGA-based signal compressor. The design facilitates more efficient use of the output capacity available to systems affected by external IO bandwidth mismatches. Despite its simple nature, the design and its components could be optimized and used as a platform for future development efforts.

The first recommendation would be to address design limitations discussed in Chapter V. In order to address issues associated with the input signal and bin frequency alignment, larger FFT periods should be utilized. This would increase the design's frequency resolution and reduce the effects of smearing in the frequency domain. In order to remove the effects of sample frequency mismatches, the design's sample frequency could be changed. Otherwise, a signal processing module could be added to incorporate interpolation and/or decimation. Additionally, the Energy (Time Window) control

module only utilizes one control algorithm, *pwr_time*, to facilitate several different processes. A more modular design could improve the overall portability of the SDR.

In order to improve the design's operational capabilities, there are three areas that are worth exploring. First, the original design concept incorporated time-frequency bins with independent time-window periods, but the final design was simplified so that each bin shared the same period. This change reduces the design's ability to evaluate diverse modulation techniques simultaneously. Additional development efforts in this area may enhance the design's real world applicability. Next, the design takes the FFT of real signal inputs. The FFT module used accommodates complex inputs. Future work could explore potential design simplification or reduced gate count by using or designing an FFT module that only uses real input signals. Finally, the SDR was designed to compress input sample data based on evaluation of the energy in each time-frequency (TF) bin. The TF construct that was implemented in the final design was based on a simple process. Some of the more sophisticated TF analysis methods mentioned in the introduction could be tested with the design's existing functional structure.

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[1] Edward Young and Paul Moakes, “DSPs vs. FPGAs for multiprocessing,” http://www.dspdesignline.com/206102177. (Accessed December 12, 2008).

[2] Chris Dick, “A Case for Using FPGAs in SDR PHY,” http://www.eetimes.com/story/OEG20020809S0049. (Accessed December 12, 2008).

[3] Jeffrey Reed, Software Radio: A Modern Approach to Radio Engineering. Prentice Hall, 2002.

[4] Antonia Papandreou-Suppappola, “Time–Frequency Processing: Tutorial on Principles and Practice,” Arizona State University.

[5] Ronald Allen, Signal analysis: time, frequency, scale, and structure, IEEE Press, 2004.

[6] Srikrishna Bhashyam, “Time-Frequency Analysis,” http://www.owlnet.rice.edu/~elec631/Projects99/mit/index2.htm . (Accessed December 12, 2008).

[7] Anwar Al-Jowder, “Analysis of digital communication signals and extraction of parameters,” Monterey, California: Naval Postgraduate School, 1994.

[8] Boualem Boashash and Peter O'shea, “Time-Frequency Analysis Applied To Signaturing Of Underwater Acoustic Signals,” University of Queensland.

[9] Amit Shoham, “FPGA Tools Bridge Gap Between Algorithm and Implementation,” http://www.dspdesignline.com/192201514. (Accessed December 12, 2008).

[10] Rodger Hosking and Richard Kuenzler, “Embedding FPGAs in DSP-driven Software Defined Radio Applications,” http://www.embedded.com/columns/technicalinsights/164302833 . (Accessed December 12, 2008).

[11] Charayaphan Charoensak, “System on Chip FPGA Design of an FM Demodulator using a Kalman Band-Pass Sigma-Delta Architecture,” Nanyang Technological University.

[12] Dick Benson, “The Design and Implementation of a GPS Receiver Channel,” DSP Magazine, October 2005.

[13] Daniel Denning , “Using System Generator To Design A Reconfigurable Video Encryption System,” Institute of System Level Integration.

[14] Konstantinos Voskakis, “Modeling and simulation of a non-coherent frequency shift keying transceiver using a Field Programmable Gate Array (FPGA),” Monterey, California: Naval Postgraduate School, 2008.

[15] Andrew La Valley, “Design and implementation of a Motor Incremental Shaft Encoder,” Monterey, California: Naval Postgraduate School, September 2008.

[16] Jingzhao Ou, “Creating Parameterized and Energy-Efficient System Generator Designs,” University of Southern California.

[17] Justin Delva, “FPGA design and verification using Simulink,” http://www.automotivedesignline.com/205800474 (Accessed December 12, 2008).

[18] Roberto Cristi, Modern digital signal processing, Thomson/Brooks/Cole, 2004.

[19] Yankin Tanurhan and Vlad Dinkevich, “DSP Design Flows in FPGAs: Strategies for designing DSP applications for FPGAs,” http://www.dspdesignline.com/187002863. (Accessed December 12, 2008).

[20] System Generator for DSP: Getting Started Guide. Xilinx Inc., November 1, 2007.

[21] ISE In-Depth Tutorial. Xilinx Inc., July 10, 2007 http://www.xilinx.com/support/techsup/tutorials/tutorials9.htm .November 14, 2007. (Accessed December 12,2008).

[22] System Generator Tutorial. Xilinx Inc., 2007.

[23] Sudhakar Yalamanchili, VHDL a Starter’s Guide, 2nd Edition. Prentice Hall, 2005.

[24] Deepak Kumar Tala, “Verilog In One Day,” http://www.asic-world.com/verilog/verilog_one_day1.html. (Accessed December 12, 2008).

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Frank Kragh
   Naval Postgraduate School
   Monterey, California

4. Herschel Loomis
   Naval Postgraduate School
   Monterey, California

5. Alan Ross
   Naval Postgraduate School
   Monterey, California

6. Roberto Cristi
   Naval Postgraduate School
   Monterey, California

7. Alexander Julian
   Naval Postgraduate School
   Monterey, California

8. Donna Miller
   Naval Postgraduate School
   Monterey, California

9. Durke Wright
   Naval Network Warfare Command
   Little Creek, Virginia